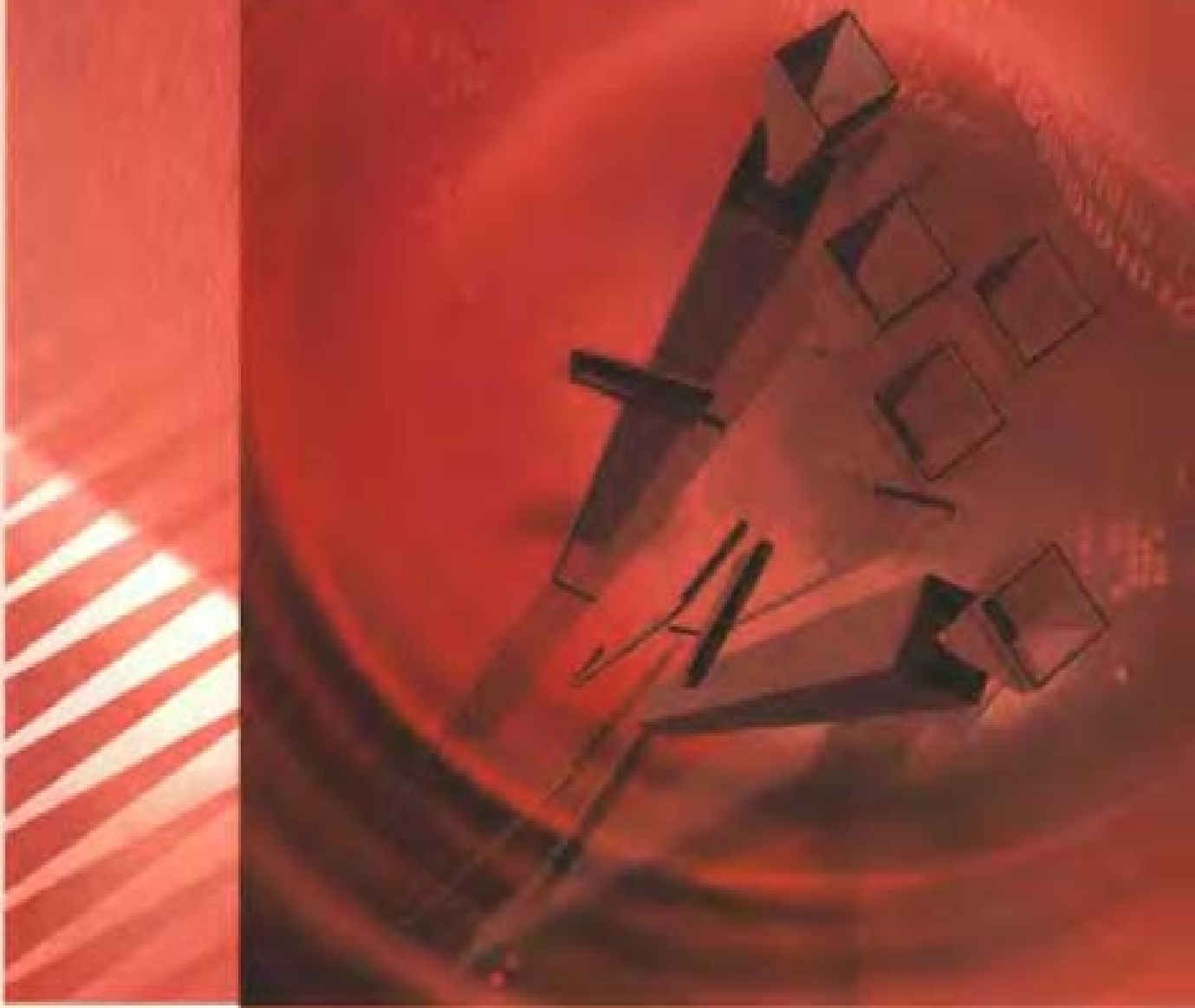
普通高等教育“十五”国家级规划教材

数据库系统概论

（第四版）

王珊 萨师煊



高等教育出版社

面向21世纪课程教材
Textbook Series for 21st Century

《数据库系统概论》第一版1987年获国家级优秀教材奖；第三版是"面向21世纪课程教材"和普通高等教育"九五"国家教委重点教材，2002年获全国普通高等学校优秀教材一等奖；第四版是普通高等教育"十五"国家级规划教材；由王珊老师主讲的数据库系统概论课程于2005年被评为国家精品课程。

ISBN 7-04-019583-6
9 787040 195835 >
定价 33.80元

普通高等教育“十五”国家级规划教材
面 向 21 世 纪 课 程 教 材

数据库系统概论

（第四版）

王 珊 萨师煊

高 等 教 育 出 版 社

内容提要

本书第1版、第2版和第3版分别于1983年、1991年、2000年出版。第3版被列为"面向21世纪课程教材",第4版是普通高等教育"十五"国家级规划教材,相应课程于2005年被评为国家精品课程。

本书系统全面地阐述数据库系统的基础理论、基本技术和基本方法。全书分为4篇17章。基础篇包括绪论、关系数据库、关系数据库标准语言SQL、数据库安全性和数据库完整性,共5章;设计与应用开发篇包括关系数据理论、数据库设计和数据库编程,共3章;系统篇包括关系查询处理和查询优化、数据库恢复技术、并发控制和数据库管理系统,共4章;新技术篇包括数据库技术新发展、分布式数据库系统、对象关系数据库系统、XML数据库和数据仓库与联机分析处理技术,共5章。

本书可以作为高等学校计算机专业、信息管理与信息系统等相关专业数据库课程的教材。也可供从事数据库系统研究、开发和应用的研究人员和工程技术人员参考。

图书在版编目(CIP)数据

数据库系统概论 / 王珊, 萨师煊. —4版. —北京: 高等教育出版社, 2006.5
ISBN 7-04-019583-6

Ⅰ.数... Ⅱ.①王...②萨... Ⅲ.数据库系统-高等学校-教材 Ⅳ.TP311.13

中国版本图书馆CIP数据核字(2006)第060254号

策划编辑 倪文慧 **责任编辑** 韩 飞 **封面设计** 于文燕 **责任印制** 韩 刚

出版发行	高等教育出版社	**购书热线**	010-58581118
社 址	北京市西城区德外大街4号	**免费咨询**	800-810-0598
邮政编码	100011	**网 址**	http://www.hep.edu.cn
总 机	010-58581000		http://www.hep.com.cn
		网上订购	http://www.landraco.com
经 销	蓝色畅想图书发行有限公司		http://www.landraco.com.cn
印 刷	北京中科印刷有限公司	**畅想教育**	http://www.widedu.com
		版 次	1983年4月第1版
开 本	787×1092 1/16		2006年5月第4版
印 张	27.25	**印 次**	2006年5月第1次印刷
字 数	570 000	**定 价**	33.80元

本书如有缺页、倒页、脱页等质量问题,请到所购图书销售部门联系调换。

版权所有 侵权必究

物料号 19583-00

作者简介



王珊教授，中国人民大学信息学院教授、博士生导师。中国计算机学会副理事长，中国计算机学会数据库专委会主任，教育部第五届科学技术委员会委员，中国科学技术协会第六届全国委员会委员等。

自1981年以来一直从事数据库方向的教学、科研和系统开发工作。1984—1986年应邀到美国马里兰大学工作，设计和开发可扩展的关系数据库管理系统XDB。回国后主持和承担了国家科技攻关项目、863高科技计划项目、北京市科技计划重大项目，国家自然科学基金重点、重大项目等30多项；在国内外杂志发表论文200余篇，出版数据库方面的著作20多部。

20多年来她主持研制和开发了多个具有自主版权的数据库管理系统。她的科研和教学成果曾获得多项国家级和省部级科技进步奖、国家级优秀教材奖和国家精品课程以及全国优秀教师和全国五一劳动奖章等荣誉称号。



萨师煊教授，我国最早从事数据库教学和科研的专家之一。是中国数据库学科的奠基人。在国内率先设立了数据库系统课程，发表了许多论文和著作。历任中国人民大学信息系主任、中国计算机学会常务理事、数据库学组组长等职务。

萨师煊教授是我国数据库学术活动的倡导者和领导者，对我国数据库技术的发展、应用和学术交流起了很大的推动作用，对我国数据库技术跟踪国际前沿、缩短我国数据库技术与国际差距做出了很大贡献。

第一版前言

数据库是数据管理的最新技术，是计算机科学的重要分支。十余年来，数据库管理系统已从专用的应用程序包发展成为通用的系统软件。由于数据库具有数据结构化、最低冗余度、较高的程序与数据独立性、易于扩充、易于编制应用程序等优点，较大的信息系统都是建立在数据库设计之上的。因此，不仅大型计算机及中小型计算机，甚至微型机都配有数据库管理系统。目前，对数据库各种模型的研究以及理论上的探讨都还在蓬勃开展，其应用也从一般管理扩大到计算机辅助设计、人工智能以及科技计算等领域。国外高等学校计算机科学系、计算机应用与信息系统系等都开设有数据库系统方面的课程。今年来，我国在计算机科学教育中也对数据库予以应有的重视。1983 年教育部部属高等学校计算机软件专业教学方案将数据库概论列为四年制本科的必修课程，并已于 1983 年 6 月通过了教学大纲，我们这本书便是按照该大纲编写的一本教材。

全书共分十章和两个附录。附录Ⅰ的“文件组织”系学习数据库的预备知识，但已散见于数据结构等先行课中，列在这里，仅供参阅用，一般不必作为正式讲授内容。第一、二、三章为有关数据库的一般概念；第四至八章介绍三种重要数据库模型。其中第八章的公理系统与模式分解的大部分内容已超出大纲范围，讲授时可酌情取舍。第九章为数据库控制；第十章为数据库设计。附录Ⅱ的“IMAGE3000 数据库系统”介绍了如何在 HP－3000 计算机的 IMAGE3000 数据库管理系统上建立数据库和使用数据库。对于能够用到 HP－3000 计算机的单位，这一部分可作为实习教材；对于不能用上 HP－3000 计算机的单位则可作为参考，借以从中了解如何在一个实际的数据库管理系统上建立一个数据库应用系统。

本书是编者根据多年来进行数据库讲演及在高等学校为研究生、本科生开设数据库系统概论课程的经验，并在多次编写的内部交流讲义的基础上修改而成的，由于我们水平不高，而数据库作为计算机科学中的一个新兴分支发展又非常迅速，因此，在本书中存在的问题一定不少，希望读者提出批评意见。

中国科技大学研究生院罗晓沛同志、清华大学计算机科学与工程系杨德元同志、俞盘祥同志与沈金发同志详细阅读了全稿，并提出许多有益的意见，在此谨向他们致以衷心的感谢。

作为内部交流讲义阶段，有 50 余所兄弟院校采用作为教本，其中不少同志在使用以后曾以口头或书面形式提出意见，我们也对他们表示感谢。

在讲稿整理过程中，我校信息管理系的刘怡同志、前 78 级、79 级研究生吴鸥琦等同志，

82级研究生张秉训、刘伶、杨明伟等同志以及资料室的王小平等同志，都从不同方面分别做了一些工作，对此作者一并表示诚挚的谢意。

萨师煊　王　珊

1983年4月

于中国人民大学

第二版前言

《数据库系统概论》第一版出版于1983年，距今已6年。在这6年中，不仅数据库技术有了很大进展，而且国内计算机专业的学生和技术人员的水平也有显著提高。因此在第二版中，我们针对这些情况对原书从结构到内容做了较大的调整、修改和增删；但原书的基本宗旨和风格不变，仍以国家教育委员会颁布的《数据库系统概论教学大纲》作为本书编写的基本依据，保持讲述数据库的基本概念、基本理论和基本技术为主的特点。

全书分为两大部分，共十一章。第一部分，第一至九章，是计算机软件专业本科生的基本教程。第二部分，第十、十一章是高级教程。

第一章绪论，概述了数据管理的进展、数据模型和数据库系统构成的一般概念。第二至七章介绍三种重要的数据库系统。它们是，网状数据库DBTG系统、层次数据库IMS和关系数据库系统。鉴于关系数据库具有许多优点并已在应用中日趋成熟，我们把重点放在这一部分，进一步充实了关系数据库的内容，共计有四章（第四至七章）。第四章概述关系模型的基本概念、关系代数和关系演算；第五章详细介绍关系数据库标准语言SQL；第六章讨论关系系统及其查询优化；第七章讲述关系数据理论。

第八章“数据库保护”（包括数据库的安全性、完整性、并发控制和恢复）和第九章“数据库设计”都作了较大的变动，充实了内容，增强了实用性。

第二部分是新增的。包括第十章“数据库管理系统”和第十一章“分布式数据库系统”。这是为了加强读者对DBMS的了解，适应对“分布处理”日益普遍的需要，引导读者从学习本书开始向某些数据库的重要新领域过渡。

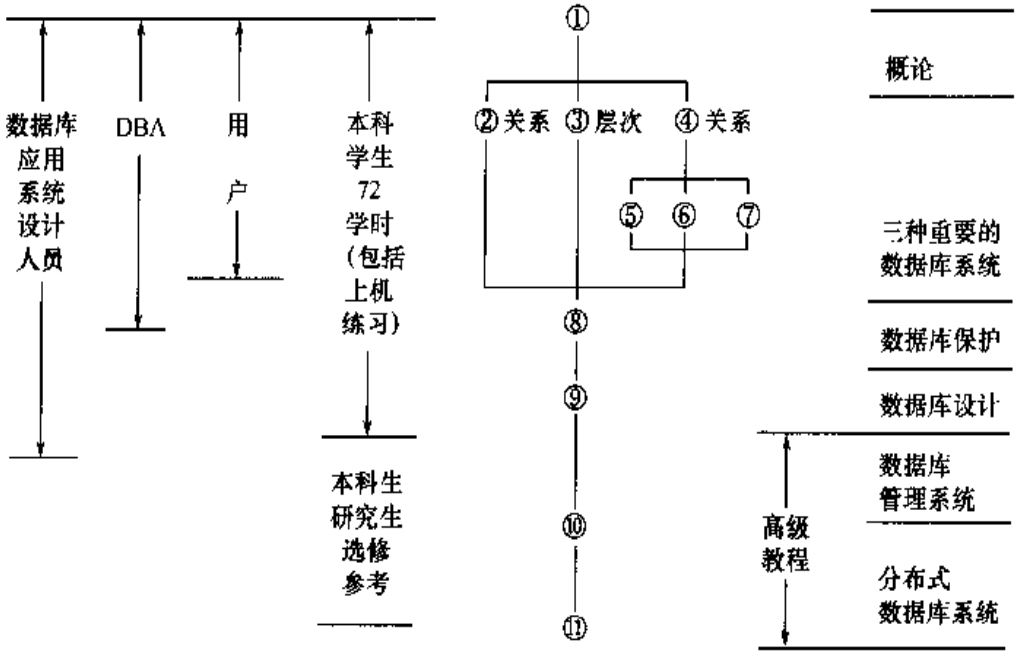
第一版中有附录Ⅰ“文件组织”与附录Ⅱ“IMAGE数据库系统”。前者是基础性教材，由于读者水平的提高已无存在的必要；后者原作为实习课的参考教材，鉴于近几年引进多种商品化的关系数据库系统且应用甚为广泛，因此我们已另行编写了一本《实用数据库系统汇编》作为实习教材（高等教育出版社出版，1990年出版），其中介绍了五种关系系统和IMAGE系统。因此本书删去了第一版中的附录。

下图给出了本书各章之间的联系和读者对象示意。

本书由王珊执笔，萨师煊审定，杜小勇、李曦老师和研究生唐元昌、武志文、任永杰等参与了内容讨论和校阅工作。

中国科技大学研究生院罗晓沛教授、清华大学俞盘祥副教授和北京大学杨冬青副教授详细审阅了全稿并提出了许多有益的意见，在此向他们致以衷心的感谢。

我们在改编过程中，尽可能引入新的观点和方法，力求能反映当代技术水平。但由于学



识浅陋，见闻不广，必有许多不足之处，希同行指正。

萨师煊　王　珊

1989 年 12 月

于中国人民大学数据工程与知识工程研究所

第三版前言

数据库技术从20世纪60年代中期产生到今天仅仅三十多年的历史，已经历了三代演变，造就了C. W. Bachman、E. F. Codd和James Gray三位图灵奖得主；发展了以数据建模和DBMS核心技术为主，内容丰富的一门学科；带动了一个巨大的软件产业——DBMS产品及其相关工具和解决方案。三十多年成就辉煌。

数据库技术是计算机科学技术中发展最快的领域之一，也是应用最广的技术之一，它已成为计算机信息系统与应用系统的核心技术和重要基础。本书系统地阐述数据库系统的理论、技术和方法，是《数据库系统概论》的第三版。第一版1983年出版（1987年获国家级优秀教材奖），第二版1991年出版。

针对数据库技术的发展和我国应用水平的提高，我们对第二版做了较大的调整、修改和增删，但原书的基本宗旨和风格不变，保持讲述数据库的基本概念、基本理论和基本技术为主的特点。第三版主要的修改是：

1. 根据我国实际情况，网状，层次数据库系统已很少使用，因此把它们删去了，有关的主要概念放在第一章数据模型中介绍。

2. 进一步加强了关系数据库系统的讲解，特别是SQL语言的介绍，以适应当前广泛使用关系数据库系统的需要。

3. 把第二版第八章数据库保护中的安全性、完整性、并发控制和恢复四节扩展为系统篇中的四章，内容作了适度的加宽和加深。随着大型数据库系统的普遍使用，这些知识和技术是运行和维护数据库系统必不可少的。本书从使用和管理的角度讲解这些知识而不是讨论实现这些功能的内部技术。

4. 为了反映数据库技术的发展，增加了新技术篇。第十二章全面介绍了数据库发展的总体轮廓，从数据模型、新技术内容、应用领域三个方面，阐述了新一代数据库系统及其相互关系。并选择当前较重要的新技术在后面三章中介绍，它们是面向对象数据库系统、分布式数据库系统和并行数据库系统。

5. 每章后配有小结，习题及阅读参考文献，并对许多文献作了简要的注释，以便读者进一步参考。

此外，为了辅助教学和加强上机实习，我们编写了与本书配套的《数据库系统概论学习指导与习题解答》，该书附有光盘。人大金仓信息技术有限公司开发了“人大金仓数据库管理系统Kingbase ES”，该系统的工作组版可以从人大金仓公司的网站http://www.kingbase.com.cn/上免费下载系统，作为你的实验平台。

全书分为四篇共十五章。第一章至第十章，是计算机软件专业本科生的基本教程（书中有＊号的部分除外），第十一章至第十五章是高级教程。

下图给出了本书各章之间的联系和读者对象示意。



本书内容丰富，讲授时可根据学生及专业情况酌情取舍。例如，对于计算机专业本科学生，第二章2.5关系演算，第五章5.4模式分解可适当压缩，新技术篇中的章节，教师可以选择部分内容进行讲解。

本书由王珊教授执笔，萨师煊教授审定，陈红副教授和研究生曹会萍、王静等参与了内容讨论和书稿校阅工作。

中国科技大学研究生院邵佩英教授和北京大学杨冬青教授详细审阅了全稿并提出了许多有益的意见，在此向他们表示衷心的感谢。

我们在编写本书的过程中，努力跟踪数据库学科的新发展、新技术，把它们纳入到教材中来，力求反映当代新技术，以保持本书的先进性和实用性。但由于学识浅陋，见闻不广，必有许多不足之处，希学术同仁不吝赐教。

萨师煊　王　珊

1999年仲夏于中国人民大学

第四版前言

数据库技术的发展十分迅速，为适应数据库技术的进展，《数据库系统概论》每隔几年就修订一次，以反映学科的新成果和应用的新方向，保持本书的先进性、科学性和实用性。

本书第1版、第2版和第3版分别于1983年、1991年、2000年出版。第4版是普通高等教育“十五”国家级规划教材与国家精品课程建设的成果。全书分为4篇17章，如下表所示。第1至第11章是本科专业的基本教程（书中有*号的部分除外），第12至第17章是高级教程。

<table>
<tr><td rowspan="4">本科生必读</td><td>1. 绪论
2. 关系数据库
3. 关系数据库标准语言SQL
4 数据库安全性
5 数据库完整性</td><td>基础篇</td><td>实验1 认识DBMS
实验2 交互式SQL
实验3 数据控制（安全性部分）
实验4 数据控制（完整性部分）</td><td rowspan="4">DBA和数据库应用系统设计开发人员</td></tr>
<tr><td>6. 关系数据理论
7 数据库设计
8. 数据库编程</td><td>设计与应用开发篇</td><td>课堂大作业《数据库设计与应用开发》
实验5 通过嵌入式SQL访问数据库
实验6 使用PL/SQL编写存储过程访问数据库
实验7 通过ODBC访问数据库
实验8 通过JDBC访问数据库</td></tr>
<tr><td>9. 关系查询处理和查询优化
10. 数据库恢复技术
11. 并发控制</td><td rowspan="2">系统篇</td><td>实验9 查询优化</td></tr>
<tr><td>12 数据库管理系统</td><td></td></tr>
<tr><td>本科生、研究生选读</td><td>13. 数据库技术新发展
14. 分布式数据库系统
15. 对象关系数据库系统
16. XML数据库
17. 数据仓库与联机分析处理技术</td><td>新技术篇</td><td></td><td></td></tr>
</table>

第4版主要修改的内容包括：

1. 基础篇中继续加强关系数据库系统的讲解，特别是SQL的内容紧跟标准的发展，更新了数据库安全性和完整性的部分内容；

2. 系统篇中重写了关系查询处理和查询优化一章，内容做了适度的加宽加深；

3. 设计与应用开发篇中增加数据库编程一章，讲解了应用开发所需要的ODBC、JDBC等概念和方法；

4. 新技术篇中修改了数据库技术新发展一章的内容，添加了反映数据库最新发展的重要技术，如对象关系数据库系统、XML 数据库、数据仓库和联机分析处理等章节，限于篇幅删去了第 3 版中并行数据库一章；

5. 最为关键的是第 4 版提供了实验环境和实验指导，进一步加强实验和课程设计等教学环节。根据教材章节的内容安排了 9 个实验和 1 个大作业。在我们的教学网站 http://www.chinadb.org 或者 http://4A.hep.com.cn 上给出了本书每个实验的详细要求和部分实验的报告示例，供读者学习参考。这些实验均使用国产金仓数据库管理系统 Kingbase ES (Kingbase Enterprise Server)作为实验平台，本书中所讲解的 SQL 例子都在 Kingbase ES 上运行通过。该系统可以从人大金仓公司的网站 http://www.Kingbase.com.cn/上免费下载。

本书内容全面丰富，教师可以针对不同专业和不同类别的学生，挑选本书中不同章节的内容进行讲解。

本书由王珊教授执笔。

中国科技大学研究生院罗晓沛教授、北京工商大学姜同强副教授详细审阅了书稿，并提出了许多有益的意见；陈红教授、杜小勇教授等根据他们讲授本书的实际体会，对内容和实验提出了修改建议；在此向他们表示衷心的感谢。

硕士研究生龚玮薇在金仓数据库 Kingbase ES 平台上运行通过了书中所有 SQL 语句，并完成了作业中的上机实验。王秋月博士、博士研究生张俊、彭朝晖、栾华、硕士研究生郑肇万、陈瑞文等参与了部分资料的收集整理工作，在此也向他们表示感谢。

在修订本书的过程中，我参考了大量国内外教材、专著、论文和资料，努力跟踪数据库学科的新发展、新技术，有选择地把它们纳入到教材中来，书中也包含了我 20 多年来教学中的经验体会和研究开发中的成果。但由于学科发展太快，本人才疏学浅，必有许多不足之处，希学术同仁不吝赐教。

王 珊

2005 年岁末于中国人民大学

目　录

第一篇　基础篇

第一章　绪论 …… (3)
1.1　数据库系统概述 …… (3)
1.1.1　数据、数据库、数据库管理系统、数据库系统 …… (3)
1.1.2　数据管理技术的产生和发展 …… (6)
1.1.3　数据库系统的特点 …… (8)
1.2　数据模型 …… (12)
1.2.1　两大类数据模型 …… (12)
1.2.2　数据模型的组成要素 …… (13)
1.2.3　概念模型 …… (14)
1.2.4　最常用的数据模型 …… (18)
1.2.5　层次模型 …… (20)
1.2.6　网状模型 …… (25)
1.2.7　关系模型 …… (28)
1.3　数据库系统结构 …… (31)
1.3.1　数据库系统模式的概念 …… (31)
1.3.2　数据库系统的三级模式结构 … (31)
1.3.3　数据库的二级映像功能与数据独立性 …… (33)
1.4　数据库系统的组成 …… (34)
1.5　小结 …… (37)
习题 …… (37)
本章参考文献 …… (39)
第二章　关系数据库 …… (41)
2.1　关系数据结构及形式化定义 …… (41)
2.1.1　关系 …… (42)
2.1.2　关系模式 …… (46)
2.1.3　关系数据库 …… (47)
2.2　关系操作 …… (47)
2.2.1　基本的关系操作 …… (47)
2.2.2　关系数据语言的分类 …… (48)
2.3　关系的完整性 …… (48)
2.3.1　关系的三类完整性约束 …… (49)
2.3.2　实体完整性(Entity Integrity) … (49)
2.3.3　参照完整性(Referential Integrity) …… (49)
2.3.4　用户定义的完整性(User-defined Integrity) …… (51)
2.4　关系代数 …… (52)
2.4.1　传统的集合运算 …… (53)
2.4.2　专门的关系运算 …… (53)
2.5　关系演算 …… (60)
2.5.1　元组关系演算语言 ALPHA …… (60)
*2.5.2　元组关系演算 …… (65)
2.5.3　域关系演算语言 QBE …… (68)
2.6　小结 …… (74)
习题 …… (74)
实验 1　认识 DBMS …… (76)
本章参考文献 …… (76)
第三章　关系数据库标准语言 SQL …… (78)
3.1　SQL 概述 …… (78)
3.1.1　SQL 的产生与发展 …… (78)
3.1.2　SQL 的特点 …… (79)
3.1.3　SQL 的基本概念 …… (81)
3.2　学生－课程数据库 …… (82)
3.3　数据定义 …… (83)
3.3.1　模式的定义与删除 …… (83)
3.3.2　基本表的定义、删除与修改 …… (84)

3.3.3 索引的建立与删除 …… (89)
3.4 数据查询 …… (91)
3.4.1 单表查询 …… (91)
3.4.2 连接查询 …… (100)
3.4.3 嵌套查询 …… (104)
3.4.4 集合查询 …… (112)
3.4.5 SELECT 语句的一般格式 …… (114)
3.5 数据更新 …… (115)
3.5.1 插入数据 …… (115)
3.5.2 修改数据 …… (117)
3.5.3 删除数据 …… (117)
3.6 视图 …… (118)
3.6.1 定义视图 …… (119)
3.6.2 查询视图 …… (121)
3.6.3 更新视图 …… (123)
3.6.4 视图的作用 …… (125)
3.7 小结 …… (127)
习题 …… (127)
实验2 交互式 SQL …… (128)
本章参考文献 …… (128)
第四章 数据库安全性 …… (130)
4.1 计算机安全性概述 …… (130)
4.1.1 计算机系统的三类安全性问题 …… (130)
4.1.2 安全标准简介 …… (131)
4.2 数据库安全性控制 …… (135)
4.2.1 用户标识与鉴别(Identification & Authentication) …… (135)
4.2.2 存取控制 …… (136)
4.2.3 自主存取控制(DAC)方法 …… (136)
4.2.4 授权(Authorization)与回收 … (137)
4.2.5 数据库角色 …… (142)
4.2.6 强制存取控制(MAC)方法 …… (143)
4.3 视图机制 …… (144)
4.4 审计(Audit) …… (145)
4.5 数据加密 …… (146)
4.6 统计数据库安全性 …… (147)
4.7 小结 …… (147)
习题 …… (148)
实验3 数据控制(安全性部分) …… (149)
本章参考文献 …… (149)
第五章 数据库完整性 …… (151)
5.1 实体完整性 …… (152)
5.1.1 实体完整性定义 …… (152)
5.1.2 实体完整性检查和违约处理 …… (153)
5.2 参照完整性 …… (154)
5.2.1 参照完整性定义 …… (154)
5.2.2 参照完整性检查和违约处理 …… (154)
5.3 用户定义的完整性 …… (156)
5.3.1 属性上的约束条件的定义 …… (156)
5.3.2 属性上的约束条件检查和违约处理 …… (157)
5.3.3 元组上的约束条件的定义 …… (158)
5.3.4 元组上的约束条件检查和违约处理 …… (158)
5.4 完整性约束命名子句 …… (158)
*5.5 域中的完整性限制 …… (160)
5.6 触发器 …… (160)
5.6.1 定义触发器 …… (160)
5.6.2 激活触发器 …… (162)
5.6.3 删除触发器 …… (163)
5.7 小结 …… (163)
习题 …… (164)
实验4 数据控制(完整性部分) …… (164)
本章参考文献 …… (165)

第二篇 设计与应用开发篇

第六章 关系数据理论 …… (169)
6.1 问题的提出 …… (169)

6.2 规范化 …… (172)
6.2.1 函数依赖 …… (172)
6.2.2 码 …… (173)
6.2.3 范式 …… (174)
6.2.4 2NF …… (175)
6.2.5 3NF …… (176)
6.2.6 BCNF …… (176)
6.2.7 多值依赖 …… (178)
6.2.8 4NF …… (181)
6.2.9 规范化小结 …… (182)
6.3 数据依赖的公理系统 …… (182)
*6.4 模式的分解 …… (187)
6.4.1 模式分解的3个定义 …… (187)
6.4.2 分解的无损连接性和保持函数依赖性 …… (188)
6.4.3 模式分解的算法 …… (191)
6.5 小结 …… (194)
习题 …… (195)
本章参考文献 …… (197)
第七章 数据库设计 …… (198)
7.1 数据库设计概述 …… (198)
7.1.1 数据库设计的特点 …… (199)
7.1.2 数据库设计方法 …… (200)
7.1.3 数据库设计的基本步骤 …… (201)
7.1.4 数据库设计过程中的各级模式 …… (203)
7.2 需求分析 …… (205)
7.2.1 需求分析的任务 …… (205)
7.2.2 需求分析的方法 …… (206)
7.2.3 数据字典 …… (207)
7.3 概念结构设计 …… (209)
7.3.1 概念结构 …… (209)
7.3.2 概念结构设计的方法与步骤 … (209)
7.3.3 数据抽象与局部视图设计 …… (212)
7.3.4 视图的集成 …… (219)
7.4 逻辑结构设计 …… (224)
7.4.1 E-R图向关系模型的转换 …… (224)
7.4.2 数据模型的优化 …… (226)
7.4.3 设计用户子模式 …… (227)
7.5 数据库的物理设计 …… (227)
7.5.1 数据库的物理设计的内容和方法 …… (228)
7.5.2 关系模式存取方法选择 …… (228)
7.5.3 确定数据库的存储结构 …… (230)
7.5.4 评价物理结构 …… (231)
7.6 数据库的实施和维护 …… (231)
7.6.1 数据的载入和应用程序的调试 …… (231)
7.6.2 数据库的试运行 …… (232)
7.6.3 数据库的运行和维护 …… (233)
7.7 小结 …… (234)
习题 …… (234)
课程大作业 《数据库设计与应用开发》…… (234)
本章参考文献 …… (235)
第八章 数据库编程 …… (237)
8.1 嵌入式SQL …… (237)
8.1.1 嵌入式SQL的处理过程 …… (237)
8.1.2 嵌入式SQL语句与主语言之间的通信 …… (238)
8.1.3 不用游标的SQL语句 …… (241)
8.1.4 使用游标的SQL语句 …… (243)
8.1.5 动态SQL …… (244)
8.1.6 小结 …… (245)
8.2 存储过程 …… (245)
8.2.1 PL/SQL的块结构 …… (245)
8.2.2 变量常量的定义 …… (245)
8.2.3 控制结构 …… (246)
8.2.4 存储过程 …… (247)
8.2.5 小结 …… (249)
8.3 ODBC编程 …… (249)
8.3.1 数据库互连概述 …… (249)
8.3.2 ODBC工作原理概述 …… (250)
8.3.3 ODBC API基础 …… (252)
8.3.4 ODBC的工作流程 …… (253)
8.3.5 小结 …… (258)
实验5 通过嵌入式SQL访问数据库 … (258)

实验6 使用PL/SQL编写存储过程访问数据库 …… (258)
实验7 通过ODBC访问数据库 …… (259)
*实验8 通过JDBC访问数据库 …… (259)
本章参考文献 …… (260)

第三篇 系 统 篇

第九章 关系查询处理和查询优化 …… (263)
9.1 关系数据库系统的查询处理 …… (263)
9.1.1 查询处理步骤 …… (263)
9.1.2 实现查询操作的算法示例 …… (264)
9.2 关系数据库系统的查询优化 …… (266)
9.2.1 查询优化概述 …… (266)
9.2.2 一个实例 …… (267)
9.3 代数优化 …… (269)
9.3.1 关系代数表达式等价变换规则 …… (269)
9.3.2 查询树的启发式优化 …… (271)
9.4 物理优化 …… (272)
9.4.1 基于启发式规则的存取路径选择优化 …… (273)
9.4.2 基于代价的优化 …… (274)
9.5 小结 …… (275)
习题 …… (275)
实验9 查询优化 …… (276)
本章参考文献 …… (276)
第十章 数据库恢复技术 …… (278)
10.1 事务的基本概念 …… (278)
10.2 数据库恢复概述 …… (279)
10.3 故障的种类 …… (280)
10.4 恢复的实现技术 …… (282)
10.4.1 数据转储 …… (282)
10.4.2 登记日志文件(Logging) …… (283)
10.5 恢复策略 …… (285)
10.5.1 事务故障的恢复 …… (285)
10.5.2 系统故障的恢复 …… (285)
10.5.3 介质故障的恢复 …… (286)
10.6 具有检查点的恢复技术 …… (286)
10.7 数据库镜像 …… (288)
10.8 小结 …… (289)
习题 …… (289)
本章参考文献 …… (290)
第十一章 并发控制 …… (292)
11.1 并发控制概述 …… (293)
11.2 封锁 …… (294)
11.3 活锁和死锁 …… (296)
11.3.1 活锁 …… (296)
11.3.2 死锁 …… (297)
11.4 并发调度的可串行性 …… (299)
11.4.1 可串行化调度 …… (299)
11.4.2 冲突可串行化调度 …… (300)
11.5 两段锁协议 …… (301)
11.6 封锁的粒度 …… (302)
11.6.1 多粒度封锁 …… (303)
11.6.2 意向锁 …… (304)
11.7 小结 …… (305)
习题 …… (305)
本章参考文献 …… (306)
***第十二章 数据库管理系统** …… (308)
12.1 DBMS的基本功能 …… (308)
12.2 DBMS的进程结构和多线索机制 … (309)
12.2.1 *N*方案:DBMS与应用程序相融合的方案 …… (309)
12.2.2 2*N*方案:一个DBMS进程对应一个用户进程 …… (310)
12.2.3 *N*+1方案:一个DBMS进程对应所有用户进程 …… (311)
12.2.4 *N*+*M*方案:*M*个DBMS进程对应*N*个用户进程 …… (312)
12.2.5 多线索(Multi_Threaded)DBMS的概念 …… (313)

12.3 DBMS系统结构 …… (316)
12.3.1 DBMS的层次结构 …… (316)
12.3.2 RDBMS的运行过程示例 …… (317)
12.4 语言处理 …… (318)
12.4.1 语言处理层的任务和工作步骤 …… (318)
12.4.2 解释方法 …… (320)
12.4.3 预编译方法 …… (321)
12.5 数据存取层 …… (322)
12.5.1 数据存取层的系统结构 …… (323)
12.5.2 数据存取层的功能子系统 … (324)
12.6 缓冲区管理 …… (326)
12.7 数据库物理组织 …… (328)
12.8 小结 …… (329)
习题 …… (330)
本章参考文献 …… (330)

第四篇 新技术篇

第十三章 数据库技术新发展 …… (335)
13.1 数据库技术发展概述 …… (335)
13.2 数据库发展的3个阶段 …… (336)
13.2.1 第一代数据库系统 …… (336)
13.2.2 第二代数据库系统 …… (337)
13.2.3 新一代数据库系统 …… (338)
13.3 数据库系统发展的特点 …… (339)
13.3.1 数据模型的发展 …… (340)
13.3.2 数据库技术与其他相关技术相结合 …… (341)
13.3.3 面向应用领域的数据库新技术 …… (343)
13.4 数据库技术发展趋势 …… (346)
13.5 小结 …… (349)
习题 …… (350)
本章参考文献 …… (350)
第十四章 分布式数据库系统 …… (352)
14.1 概述 …… (352)
14.1.1 分布式数据库系统 …… (352)
14.1.2 分布式数据库系统的特点 … (354)
14.2 分布式数据库系统的体系结构 … (356)
14.2.1 分布式数据库系统的模式结构 …… (356)
14.2.2 数据分片 …… (357)
14.2.3 分布透明性 …… (359)
14.2.4 分布式数据库管理系统 …… (360)
14.3 查询处理和优化 …… (362)
14.3.1 一个实例 …… (362)
14.3.2 查询处理和优化要解决的问题 …… (364)
14.3.3 查询优化的目标 …… (365)
14.3.4 连接查询的优化 …… (366)
14.4 分布事务管理 …… (368)
14.4.1 分布事务的恢复 …… (368)
14.4.2 并发控制 …… (369)
14.5 小结 …… (370)
习题 …… (371)
本章参考文献 …… (372)
第十五章 对象关系数据库系统 …… (374)
15.1 面向对象数据模型 …… (375)
15.1.1 OO模型的核心概念 …… (375)
15.1.2 类层次(结构) …… (376)
15.1.3 继承 …… (377)
15.1.4 对象的嵌套 …… (377)
15.2 对象-关系数据库 …… (379)
15.2.1 对象关系数据库系统中扩展的关系数据类型 …… (379)
15.2.2 对象关系数据库系统中扩展的对象类型及其定义 …… (381)
15.2.3 参照类型(Reference Type) … (383)
15.2.4 继承性 …… (385)
15.2.5 子表和超表 …… (385)
15.3 小结 …… (387)
习题 …… (388)

本章参考文献 …… (388)
第十六章 XML 数据库 …… (391)
16.1 概述 …… (391)
16.2 XML 简介 …… (392)
16.3 XML 数据模型 …… (394)
16.3.1 XML DTD …… (396)
16.3.2 XML Schema …… (397)
16.4 SQL/XML …… (398)
16.4.1 XML 数据类型 …… (399)
16.4.2 XML 发布函数 …… (400)
16.4.3 XML 提取函数 …… (405)
16.5 小结 …… (406)
习题 …… (407)
本章参考文献 …… (407)
第十七章 数据仓库与联机分析处理技术 …… (408)
17.1 数据仓库技术 …… (408)
17.2 联机分析处理技术 …… (412)
17.3 数据挖掘技术 …… (414)
17.4 小结 …… (416)
习题 …… (417)
本章参考文献 …… (417)

第一篇

基　础　篇

本篇讲解数据库系统的基本概念和基础知识，是读者进一步学习下面各个章节以及数据库系统其他课程的基础。

基础篇包括5章。

第一章绪论，初步讲解数据库的基本概念，介绍数据模型的组成要素和最主要的数据模型、数据库系统的三级模式结构和数据库系统的主要组成部分。

第二章关系数据库，系统讲解关系数据库的重要概念，包括关系模型和关系代数。

第三章关系数据库标准语言SQL，系统而详尽地讲解SQL的数据定义、数据查询、数据更新三部分功能。

第四、第五章讲解关系数据库的数据控制功能。

第四章数据库安全性，全面讲解实现数据库系统安全性的技术和方法，着重讲解SQL存取控制功能，包括GRANT和REVOKE语句、视图机制、审计功能等。

第五章数据库完整性，系统而详尽地讲解实体完整性、参照完整性和用户定义的完整性。包括这些完整性约束的定义方法、完整性检查机制和违约处理，讲解触发器的用处和使用方法。

第一章 绪 论

数据库是数据管理的最新技术，是计算机科学的重要分支。今天，信息资源已成为各个部门的重要财富和资源。建立一个满足各级部门信息处理要求的行之有效的信息系统也成为一个企业或组织生存和发展的重要条件。因此，作为信息系统核心和基础的数据库技术得到越来越广泛的应用，从小型单项事务处理系统到大型信息系统，从联机事务处理（On-Line Transaction Processing，OLTP）到联机分析处理（On-Line Analysis Processing，OLAP），从一般企业管理到计算机辅助设计与制造（CAD/CAM）、计算机集成制造系统（CIMS）、电子政务（e-Commerce）、电子商务（e-Business）地理信息系统（GIS）等，越来越多新的应用领域采用数据库技术来存储和处理信息资源。

对于一个国家来说，数据库的建设规模、数据库信息量的大小和使用频度已成为衡量这个国家信息化程度的重要标志。因此，数据库课程不仅是计算机科学与技术专业、信息管理专业的重要课程，也是许多非计算机专业的选修课程。

本章介绍数据库系统的基本概念，包括数据管理的发展过程、数据库系统的组成部分等。读者从中可以学习到为什么要使用数据库技术、数据库技术的重要性。本章是后面各章节的准备和基础。

1.1 数据库系统概述

在系统地介绍数据库的基本概念之前，这里首先介绍一些数据库最常用的术语和基本概念。

1.1.1 数据、数据库、数据库管理系统、数据库系统

数据、数据库、数据库管理系统和数据库系统是与数据库技术密切相关的4个基本概念。

一、数据（Data）

数据是数据库中存储的基本对象。数据在大多数人头脑中的第一个反映就是数字，例

如93、1 000、99.5、-330.86、¥6880、$726等。其实数字只是最简单的一种数据，是数据的一种传统和狭义的理解。广义的理解，数据的种类很多，文本(text)、图形(graph)、图像(image)、音频(audio)、视频(video)、学生的档案记录、货物的运输情况等，这些都是数据。

可以对数据做如下定义：描述事物的符号记录称为数据。描述事物的符号可以是数字，也可以是文字、图形、图像、声音、语言等，数据有多种表现形式，它们都可以经过数字化后存入计算机。

在现代计算机系统中数据的概念是广义的。早期的计算机系统主要用于科学计算，处理的数据是数值型数据，如整数、实数、浮点数等。现在计算机存储和处理的对象十分广泛，表示这些对象的数据就也越来越复杂了。

数据的表现形式还不能完全表达其内容，需要经过解释，数据和关于数据的解释是不可分的。例如，93是一个数据，可以是一个同学某门课的成绩，也可以是某个人的体重，还可以是计算机系2003级的学生人数。数据的解释是指对数据含义的说明，数据的含义称为数据的语义，数据与其语义是不可分的。

在日常生活中，人们可以直接用自然语言(如汉语)来描述事物。例如，可以这样来描述某校计算机系一位同学的基本情况：李明同学，男，1972年5月生，江苏省南京市人，1990年入学。在计算机中常常这样来描述：

(李明，男，197205，江苏省南京市，计算机系，1990)

即把学生的姓名、性别、出生年月、出生地、所在院系、入学时间等组织在一起，组成一个记录。这里的学生记录就是描述学生的数据。这样的数据是有结构的。记录是计算机中表示和存储数据的一种格式或一种方法。

二、数据库(DataBase，简称DB)

数据库，顾名思义，是存放数据的仓库。只不过这个仓库是在计算机存储设备上，而且数据是按一定的格式存放的。

人们收集并抽取出一个应用所需要的大量数据之后，应将其保存起来，以供进一步加工处理，进一步抽取有用信息。在科学技术飞速发展的今天，人们的视野越来越广，数据量急剧增加。过去人们把数据存放在文件柜里，现在人们借助计算机和数据库技术科学地保存和管理大量的复杂的数据，以便能方便而充分地利用这些宝贵的信息资源。

严格地讲，数据库是长期储存在计算机内、有组织的、可共享的大量数据的集合。数据库中的数据按一定的数据模型组织、描述和储存，具有较小的冗余度、较高的数据独立性和易扩展性，并可为各种用户共享。

概括地讲，数据库数据具有永久存储、有组织和可共享三个基本特点。

三、数据库管理系统(DataBase Management System，DBMS)

了解了数据和数据库的概念，下一个问题就是如何科学地组织和存储数据，如何高效地获取和维护数据。完成这个任务的是一个系统软件——数据库管理系统。

数据库管理系统是位于用户与操作系统之间的一层数据管理软件。数据库管理系统和操作系统一样是计算机的基础软件，也是一个大型复杂的软件系统。它的主要功能包括以下几个方面：

1. 数据定义功能

DBMS 提供数据定义语言(Data Definition Language，DDL)，用户通过它可以方便地对数据库中的数据对象进行定义。

2. 数据组织、存储和管理

DBMS 要分类组织、存储和管理各种数据，包括数据字典、用户数据、数据的存取路径等。要确定以何种文件结构和存取方式在存储级上组织这些数据，如何实现数据之间的联系。数据组织和存储的基本目标是提高存储空间利用率和方便存取，提供多种存取方法(如索引查找、Hash 查找、顺序查找等)来提高存取效率。

3. 数据操纵功能

DBMS 还提供数据操纵语言(Data Manipulation Language，DML)，用户可以使用 DML 操纵数据，实现对数据库的基本操作，如查询、插入、删除和修改等。

4. 数据库的事务管理和运行管理

数据库在建立、运用和维护时由数据库管理系统统一管理、统一控制，以保证数据的安全性、完整性、多用户对数据的并发使用及发生故障后的系统恢复。

5. 数据库的建立和维护功能

它包括：数据库初始数据的输入、转换功能，数据库的转储、恢复功能，数据库的重组织功能和性能监视、分析功能等。这些功能通常是由一些实用程序或管理工具完成的。

6. 其他功能

包括：DBMS 与网络中其他软件系统的通信功能；一个 DBMS 与另一个 DBMS 或文件系统的数据转换功能；异构数据库之间的互访和互操作功能等。

数据库管理系统是数据库系统的一个重要组成部分。

四、数据库系统(DataBase System，DBS)

数据库系统是指在计算机系统中引入数据库后的系统，一般由数据库、数据库管理系统(及其开发工具)、应用系统、数据库管理员构成。应当指出的是，数据库的建立、使用和维护等工作只靠一个 DBMS 远远不够，还要有专门的人员来完成，这些人被称为数据库管理员(DataBase Administrator，DBA)。

在一般不引起混淆的情况下常常把数据库系统简称为数据库。

数据库系统可以用图 1.1 表示。数据库系统在整个计算机系统中的地位如图 1.2 所示。

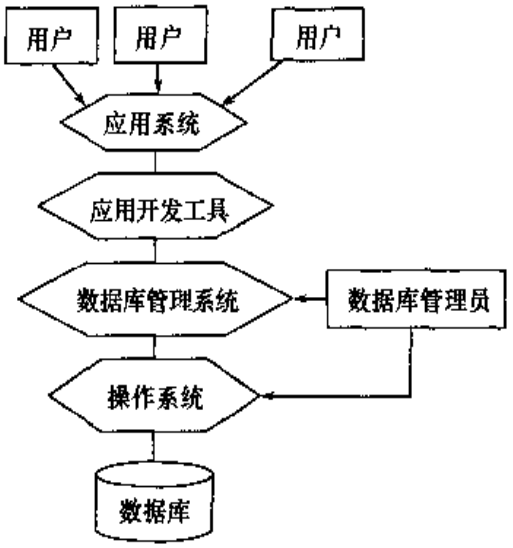


图 1.1 数据库系统

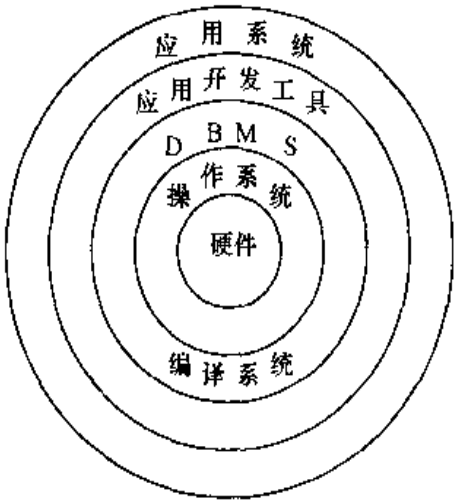


图 1.2 数据库在计算机系统中的地位

1.1.2 数据管理技术的产生和发展

数据库技术是应数据管理任务的需要而产生的。

数据管理则是指对数据进行分类、组织、编码、存储、检索和维护，它是数据处理的中心问题。而数据的处理是指对各种数据进行收集、存储、加工和传播的一系列活动的总和。

在应用需求的推动下，在计算机硬件、软件发展的基础上，数据管理技术经历了人工管理、文件系统、数据库系统 3 个阶段。这 3 个阶段的特点及其比较如表 1.1 所示。

表 1.1 数据管理 3 个阶段的比较

		人工管理阶段	文件系统阶段	数据库系统阶段
背景	应用背景	科学计算	科学计算、数据管理	大规模数据管理
	硬件背景	无直接存取存储设备	磁盘、磁鼓	大容量磁盘、磁盘阵列
	软件背景	没有操作系统	有文件系统	有数据库管理系统
	处理方式	批处理	联机实时处理、批处理	联机实时处理、分布处理、批处理
特点	数据的管理者	用户(程序员)	文件系统	数据库管理系统
	数据面向的对象	某一应用程序	某一应用	现实世界(一个部门、企业、跨国组织等)
	数据的共享程度	无共享，冗余度极大	共享性差，冗余度大	共享性高，冗余度小
	数据的独立性	不独立，完全依赖于程序	独立性差	具有高度的物理独立性和一定的逻辑独立性
	数据的结构化	无结构	记录内有结构、整体无结构	整体结构化，用数据模型描述
	数据控制能力	应用程序自己控制	应用程序自己控制	由数据库管理系统提供数据安全性、完整性、并发控制和恢复能力

一、人工管理阶段

20世纪50年代中期以前，计算机主要用于科学计算。当时的硬件状况是，外存只有纸带、卡片、磁带，没有磁盘等直接存取的存储设备；软件状况是，没有操作系统，没有管理数据的专门软件；数据处理方式是批处理。人工管理数据具有如下特点：

(1) 数据不保存

由于当时计算机主要用于科学计算，一般不需要将数据长期保存，只是在计算某一课题时将数据输入，用完就撤走。不仅对用户数据如此处置，对系统软件有时也是这样。

(2) 应用程序管理数据

数据需要由应用程序自己设计、说明(定义)和管理，没有相应的软件系统负责数据的管理工作。应用程序中不仅要规定数据的逻辑结构，而且要设计物理结构，包括存储结构、存取方法、输入方式等。因此程序员负担很重。

(3) 数据不共享

数据是面向应用程序的，一组数据只能对应一个程序。当多个应用程序涉及某些相同的数据时，必须各自定义，无法互相利用、互相参照，因此程序与程序之间有大量的冗余数据。

(4) 数据不具有独立性

数据的逻辑结构或物理结构发生变化后，必须对应用程序做相应的修改，这就加重了程序员的负担。

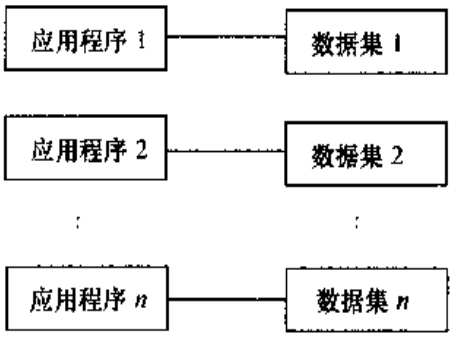


图 1.3 人工管理阶段应用程序与数据之间的对应关系

在人工管理阶段，程序与数据之间的一一对应关系可用图1.3表示。

二、文件系统阶段

20世纪50年代后期到60年代中期，这时硬件方面已有了磁盘、磁鼓等直接存取存储设备；软件方面，操作系统中已经有了专门的数据管理软件，一般称为文件系统；处理方式上不仅有了批处理，而且能够联机实时处理。

用文件系统管理数据具有如下特点：

(1) 数据可以长期保存

由于计算机大量用于数据处理，数据需要长期保留在外存上反复进行查询、修改、插入和删除等操作。

(2) 由文件系统管理数据

由专门的软件即文件系统进行数据管理，文件系统把数据组织成相互独立的数据文件，利用“按文件名访问，按记录进行存取”的管理技术，可以对文件进行修改、插入和删除的操作。文件系统实现了记录内的结构性，但整体无结构。程序和数据之间由文件系统提供存取方法进行转换，使应用程序与数据之间有了一定的独立性，程序员可以不必过多地考虑物

理细节,将精力集中于算法。而且数据在存储上的改变不一定反映在程序上,大大节省了维护程序的工作量。

但是,文件系统仍存在以下缺点:

(1) 数据共享性差,冗余度大

在文件系统中,一个(或一组)文件基本上对应于一个应用程序,即文件仍然是面向应用的。当不同的应用程序具有部分相同的数据时,也必须建立各自的文件,而不能共享相同的数据,因此数据的冗余度大,浪费存储空间。同时由于相同数据的重复存储、各自管理,容易造成数据的不一致性,给数据的修改和维护带来了困难。

(2) 数据独立性差

文件系统中的文件是为某一特定应用服务的,文件的逻辑结构对该应用程序来说是优化的,因此要想对现有的数据再增加一些新的应用会很困难,系统不容易扩充。

一旦数据的逻辑结构改变,必须修改应用程序,修改文件结构的定义。应用程序的改变,例如应用程序改用不同的高级语言编写,也将引起文件数据结构的改变。因此数据与程序之间仍缺乏独立性。可见,文件系统仍然是一个不具有弹性的无结构的数据集合,即文件之间是孤立的,不能反映现实世界事物之间的内在联系。

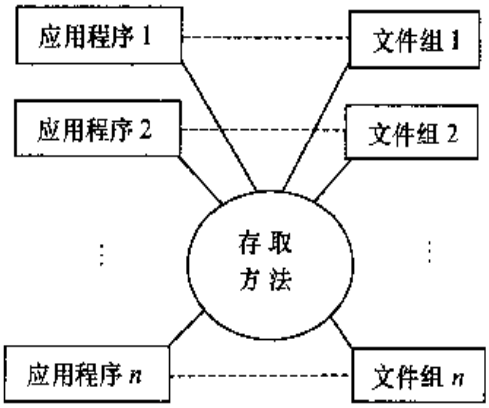


图 1.4 文件系统阶段应用程序与数据之间的对应关系

文件系统阶段程序与数据之间的关系如图 1.4 所示。

三、数据库系统阶段

20 世纪 60 年代后期以来,计算机管理的对象规模越来越大,应用范围越来越广泛,数据量急剧增长,同时多种应用、多种语言互相覆盖地共享数据集合的要求越来越强烈。

这时硬件已有大容量磁盘,硬件价格下降;软件则价格上升,为编制和维护系统软件及应用程序所需的成本相对增加;在处理方式上,联机实时处理要求更多,并开始提出和考虑分布处理。在这种背景下,以文件系统作为数据管理手段已经不能满足应用的需求,于是为解决多用户、多应用共享数据的需求,使数据为尽可能多的应用服务,数据库技术便应运而生,出现了统一管理数据的专门软件系统——数据库管理系统。

用数据库系统来管理数据比文件系统具有明显的优点,从文件系统到数据库系统,标志着数据管理技术的飞跃。下面来详细地讨论数据库系统的特点及其带来的优点。

1.1.3 数据库系统的特点

与人工管理和文件系统相比,数据库系统的特点主要有以下几个方面。

一、数据结构化

数据库系统实现整体数据的结构化，这是数据库的主要特征之一，也是数据库系统与文件系统的本质区别。

所谓"整体"结构化是指在数据库中的数据不再仅仅针对某一个应用，而是面向全组织；不仅数据内部是结构化的，而且整体是结构化的，数据之间是具有联系的。

在文件系统中每个文件内部是有结构的，即文件由记录构成，每个记录由若干属性组成。例如，学生文件 student 的记录是由学生编号、姓名、性别、年龄、系、家庭住址、联系电话等属性组成；课程文件 course 和学生选课文件 sc 的结构见图 1.5 所示。

学生文件 student 的记录结构

学生编号	姓名	性别	年龄	系	家庭住址	联系电话

课程文件 course 的记录结构

课程编号	课程名称	学时数	教材名称

学生选课文件 sc 的记录结构

学生编号	课程编号	学期	成绩

图 1.5 学生、课程、学生选课文件结构

在文件系统中，尽管其记录内部已有了某些结构，但记录之间没有联系。例如，学生文件 student、课程文件 course 和学生选课文件 sc 是独立的 3 个文件，但实际上这 3 个文件的记录之间是有联系的，sc 的学生编号必须是 student 文件中某个学生的学号，sc 的课程编号必须是 course 文件中某门课程的编号。

在关系数据库中，关系表的记录之间的这种联系是可以用参照完整性（将在第二章中详细讲解）来表述的。如果向 sc 中增加一个学生的考试成绩，但是这个学生并没有出现在 student 关系中，关系数据库管理系统（Relational DataBase Management System，RDBMS）将拒绝执行这样的插入操作，从而保证了数据的正确性。而在文件系统中要做到这一点，必须由程序员编写一段代码在应用程序中实现。

在数据库系统中实现了整体数据的结构化。也就是说，不仅要考虑某个应用的数据结构，还要考虑整个组织的数据结构。例如，一个学校的信息系统中不仅要考虑教务处的学生学籍管理、选课管理，还要考虑学生处的学生人事管理，同时还要考虑研究生院的研究生管理、人事处的教员人事管理、科研处的科研管理等。因此，学校信息系统中的学生数据就要面向各个处室的应用而不仅仅是教务处的一个学生选课应用。可以按照图 1.6 方式为该校的信息系统组织其中的学生数据：

这种数据组织方式为各部门的应用提供了必要的记录，使整体数据结构化了。这就要求在描述数据时不仅要描述数据本身，还要描述数据之间的联系。

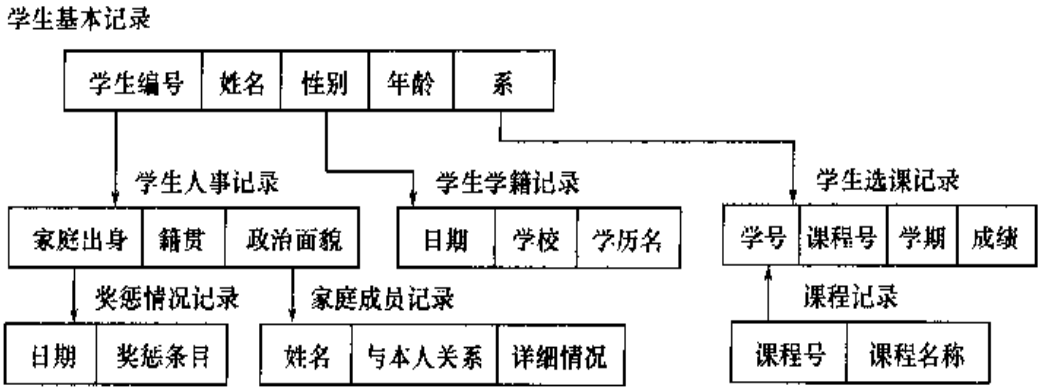


图 1.6 某学校信息系统中的学生数据

在数据库系统中,不仅数据是整体结构化的,而且存取数据的方式也很灵活,可以存取数据库中的某一个数据项、一组数据项、一个记录或一组记录。而在文件系统中,数据的存取单位是记录,粒度不能细到数据项。

二、数据的共享性高,冗余度低,易扩充

数据库系统从整体角度看待和描述数据,数据不再面向某个应用而是面向整个系统,因此数据可以被多个用户、多个应用共享使用。数据共享可以大大减少数据冗余,节约存储空间。数据共享还能够避免数据之间的不相容性与不一致性。

所谓数据的不一致性是指同一数据不同拷贝的值不一样。采用人工管理或文件系统管理时,由于数据被重复存储,当不同的应用使用和修改不同的拷贝时就很容易造成数据的不一致。在数据库中数据共享,减少了由于数据冗余造成的不一致现象。

由于数据面向整个系统,是有结构的数据,不仅可以被多个应用共享使用,而且容易增加新的应用,这就使得数据库系统弹性大,易于扩充,可以适应各种用户的要求。可以选取整体数据的各种子集用于不同的应用系统,当应用需求改变或增加时,只要重新选取不同的子集或加上一部分数据,便可以满足新的需求。

三、数据独立性高

数据独立性是数据库领域中一个常用术语和重要概念,包括数据的物理独立性和数据的逻辑独立性。

物理独立性是指用户的应用程序与存储在磁盘上的数据库中数据是相互独立的。也就是说,数据在磁盘上的数据库中怎样存储是由 DBMS 管理的,用户程序不需要了解,应用程序要处理的只是数据的逻辑结构,这样当数据的物理存储改变时,应用程序不用改变。

逻辑独立性是指用户的应用程序与数据库的逻辑结构是相互独立的,也就是说,数据的逻辑结构改变了,用户程序也可以不变。

数据独立性是由 DBMS 的二级映像功能来保证的,将在下面讨论。

数据与程序的独立,把数据的定义从程序中分离出去,加上存取数据的方法又由 DBMS 负责提供,从而简化了应用程序的编制,大大减少了应用程序的维护和修改。

四、数据由 DBMS 统一管理和控制

数据库的共享是并发的(Concurrency)共享,即多个用户可以同时存取数据库中的数据甚至可以同时存取数据库中同一个数据。

为此,DBMS 还必须提供以下几方面的数据控制功能:

(1) 数据的安全性(Security)保护

数据的安全性是指保护数据,以防止不合法的使用造成的数据的泄密和破坏。使每个用户只能按规定,对某些数据以某些方式进行使用和处理。

(2) 数据的完整性(Integrity)检查

数据的完整性指数据的正确性、有效性和相容性。完整性检查将数据控制在有效的范围内,或保证数据之间满足一定的关系。

(3) 并发(Concurrency)控制

当多个用户的并发进程同时存取、修改数据库时,可能会发生相互干扰而得到错误的结果或使得数据库的完整性遭到破坏,因此必须对多用户的并发操作加以控制和协调。

(4) 数据库恢复(Recovery)

计算机系统的硬件故障、软件故障、操作员的失误以及故意的破坏也会影响数据库中数据的正确性,甚至造成数据库部分或全部数据的丢失。DBMS 必须具有将数据库从错误状态恢复到某一已知的正确状态(亦称为完整状态或一致状态)的功能,这就是数据库的恢复功能。

数据库管理阶段应用程序与数据之间的对应关系可用图 1.7 表示。

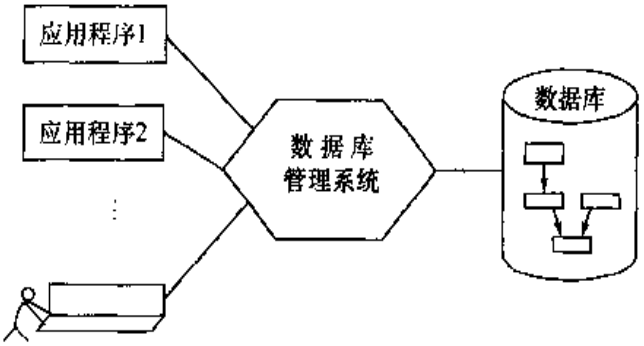


图 1.7 数据库系统阶段应用程序与数据之间的对应关系

综上所述,数据库是长期存储在计算机内有组织的大量的共享的数据集合。它可以供各种用户共享,具有最小冗余度和较高的数据独立性。DBMS 在数据库建立、运用和维护时对数据库进行统一控制,以保证数据的完整性、安全性,并在多用户同时使用数据库时进行并发控制,在发生故障后对数据库进行恢复。

数据库系统的出现使信息系统从以加工数据的程序为中心转向围绕共享的数据库为中心的新阶段。这样既便于数据的集中管理,又有利于应用程序的研制和维护,提高了数据的

利用率和相容性,提高了决策的可靠性。

目前,数据库已经成为现代信息系统的重要组成部分。具有数百 G、数百 T、甚至数百 P 字节的数据库已经普遍存在于科学技术、工业、农业、商业、服务业和政府部门的信息系统中。

数据库技术是计算机领域中发展最快的技术之一。数据库技术的发展是沿着数据模型的主线展开的。下面讨论数据模型。

1.2 数据模型

模型,特别是具体模型,人们并不陌生。一张地图、一组建筑设计沙盘、一架精致的航模飞机都是具体的模型。一眼望去,就会使人联想到真实生活中的事物。模型是对现实世界中某个对象特征的模拟和抽象。

数据模型(Data Model)也是一种模型,它是现实世界数据特征的抽象。

由于计算机不可能直接处理现实世界中的具体事物,所以人们必须事先把具体事物转换成计算机能够处理的数据。也就是首先要数字化,把现实世界中具体的人、物、活动、概念用数据模型这个工具来抽象、表示和处理。

通俗地讲,数据模型就是现实世界的模拟。

现有的数据库系统均是基于某种数据模型的。数据模型是数据库系统的核心和基础。因此,了解数据模型的基本概念是学习数据库的基础。

1.2.1 两类数据模型

数据模型应满足三方面要求:一是能比较真实地模拟现实世界;二是容易为人所理解;三是便于在计算机上实现。一种数据模型要很好地全面地满足这三方面的要求在目前尚很困难。因此,在数据库系统中针对不同的使用对象和应用目的,采用不同的数据模型。

如同在建筑设计和施工的不同阶段需要不同的图纸一样,在开发实施数据库应用系统中也需要使用不同的数据模型:概念模型、逻辑模型和物理模型。

根据模型应用的不同目的,可以将这些模型划分为两类,它们分别属于两个不同的层次。第一类是概念模型,第二类是逻辑模型和物理模型。

第一类概念模型(Conceptual Model),也称信息模型,它是按用户的观点来对数据和信息建模,主要用于数据库设计。

第二类中的逻辑模型主要包括层次模型(Hierarchical Model)、网状模型(Network Model)、关系模型(Relational Model)、面向对象模型(Object Oriented Model)和对象关系模型(Object Relational Model)等。它是按计算机系统的观点对数据建模,主要用于 DBMS 的实现。

第二类中的物理模型是对数据最低层的抽象，它描述数据在系统内部的表示方式和存取方法，在磁盘或磁带上的存储方式和存取方法，是面向计算机系统的。物理模型的具体实现是 DBMS 的任务，数据库设计人员要了解和选择物理模型，一般用户则不必考虑物理级的细节。

数据模型是数据库系统的核心和基础。各种机器上实现的 DBMS 软件都是基于某种数据模型或者说是支持某种数据模型的。

为了把现实世界中的具体事物抽象、组织为某一 DBMS 支持的数据模型，人们常常首先将现实世界抽象为信息世界，然后将信息世界转换为机器世界。也就是说，首先把现实世界中的客观对象抽象为某一种信息结构，这种信息结构并不依赖于具体的计算机系统，不是某一个 DBMS 支持的数据模型，而是概念级的模型；然后再把概念模型转换为计算机上某一 DBMS 支持的数据模型，这一过程如图 1.8 所示。



图 1.8 现实世界中客观对象的抽象过程

从现实世界到概念模型的转换是由数据库设计人员完成的，从概念模型到逻辑模型的转换可以由数据库设计人员完成，也可以用数据库设计工具协助设计人员完成，从逻辑模型到物理模型的转换一般是由 DBMS 完成的。

下面首先介绍数据模型的共性，即数据模型的组成要素，然后分别介绍两类不同的数据模型——概念模型和逻辑模型。

1.2.2 数据模型的组成要素

一般地讲，数据模型是严格定义的一组概念的集合。这些概念精确地描述了系统的静态特性、动态特性和完整性约束条件。因此数据模型通常由数据结构、数据操作和完整性约束三部分组成。

一、数据结构

数据结构描述数据库的组成对象以及对象之间的联系。也就是说，数据结构描述的内容有两类：

一类是与对象的类型、内容、性质有关的，例如，网状模型中的数据项、记录，关系模型中的域、属性、关系等；

一类是与数据之间联系有关的对象，例如网状模型中的系型(Set Type)。

数据结构是刻画一个数据模型性质最重要的方面。因此在数据库系统中，人们通常按

照其数据结构的类型来命名数据模型。例如层次结构、网状结构和关系结构的数据模型分别命名为层次模型、网状模型和关系模型。

总之,数据结构是所描述的对象类型的集合,是对系统静态特性的描述。

二、数据操作

数据操作是指对数据库中各种对象(型)的实例(值)允许执行的操作的集合,包括操作及有关的操作规则。

数据库主要有查询和更新(包括插入、删除、修改)两大类操作。数据模型必须定义这些操作的确切含义、操作符号、操作规则(如优先级)以及实现操作的语言。

数据操作是对系统动态特性的描述。

三、数据的完整性约束条件

数据的完整性约束条件是一组完整性规则。完整性规则是给定的数据模型中数据及其联系所具有的制约和依存规则,用以限定符合数据模型的数据库状态以及状态的变化,以保证数据的正确、有效、相容。

数据模型应该反映和规定本数据模型必须遵守的基本的通用的完整性约束条件。例如,在关系模型中,任何关系必须满足实体完整性和参照完整性两个条件(在关系数据库和数据库完整性等有关章节中将详细讨论这两个完整性约束条件)。

此外,数据模型还应该提供定义完整性约束条件的机制,以反映具体应用所涉及的数据必须遵守的特定的语义约束条件。例如,在某大学的数据库中规定学生成绩如果有 6 门以上不及格将不能授予学士学位,教授的退休年龄是 65 周岁,男职工的退休年龄是 60 周岁,女职工的退休年龄是 55 周岁等。

1.2.3 概念模型

由图 1.8 可以看出,概念模型实际上是现实世界到机器世界的一个中间层次。

概念模型用于信息世界的建模,是现实世界到信息世界的第一层抽象,是数据库设计人员进行数据库设计的有力工具,也是数据库设计人员和用户之间进行交流的语言,因此概念模型一方面应该具有较强的语义表达能力,能够方便、直接地表达应用中的各种语义知识,另一方面它还应该简单、清晰、易于用户理解。

一、信息世界中的基本概念

信息世界涉及的概念主要有:

(1) 实体(Entity)

客观存在并可相互区别的事物称为实体。实体可以是具体的人、事、物,也可以是抽象的概念或联系,例如,一个职工、一个学生、一个部门、一门课、学生的一次选课、部门的一次订货、老师与院系的工作关系(即某位老师在某院系工作)等都是实体。

(2) 属性(Attribute)

实体所具有的某一特性称为属性。一个实体可以由若干个属性来刻画。例如,学生实体可以由学号、姓名、性别、出生年月、所在院系、入学时间等属性组成。(94002268,张山,男,197605,计算机系,1994)这些属性组合起来表征了一个学生。

(3) 码(Key)

唯一标识实体的属性集称为码。例如学号是学生实体的码。

(4) 域(Domain)

属性的取值范围称为该属性的域。例如,学号的域为8位整数,姓名的域为字符串集合,学生年龄的域为整数,性别的域为(男,女)。

(5) 实体型(Entity Type)

具有相同属性的实体必然具有共同的特征和性质。用实体名及其属性名集合来抽象和刻画同类实体,称为实体型。例如,学生(学号,姓名,性别,出生年月,所在院系,入学时间)就是一个实体型。

(6) 实体集(Entity Set)

同一类型实体的集合称为实体集。例如,全体学生就是一个实体集。

(7) 联系(Relationship)

在现实世界中,事物内部以及事物之间是有联系的,这些联系在信息世界中反映为实体(型)内部的联系和实体(型)之间的联系。实体内部的联系通常是指组成实体的各属性之间的联系;实体之间的联系通常是指不同实体集之间的联系。

二、两个实体型之间的联系

两个实体型之间的联系可以分为三种:

① 一对一联系(1:1)

如果对于实体集 A 中的每一个实体,实体集 B 中至多有一个(也可以没有)实体与之联系,反之亦然,则称实体集 A 与实体集 B 具有一对一联系,记为1:1。

例如,学校里面,一个班级只有一个正班长,而一个班长只在一个班中任职,则班级与班长之间具有一对一联系。

② 一对多联系(1:n)

如果对于实体集 A 中的每一个实体,实体集 B 中有 n 个实体($n\geq 0$)与之联系,反之,对于实体集 B 中的每一个实体,实体集 A 中至多只有一个实体与之联系,则称实体集 A 与实体集 B 有一对多联系,记为1:n。

例如,一个班级中有若干名学生,而每个学生只在一个班级中学习,则班级与学生之间具有一对多联系。

③ 多对多联系(m:n)

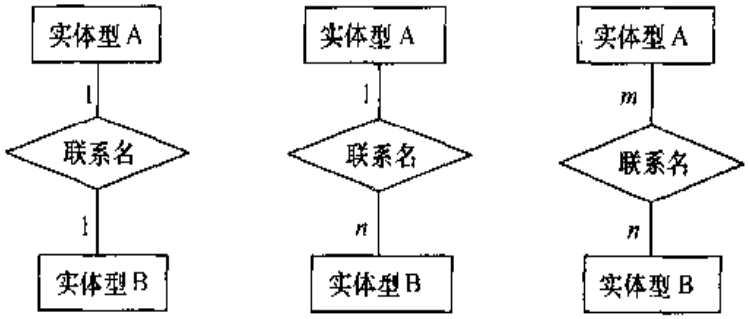
如果对于实体集 A 中的每一个实体,实体集 B 中有 n 个实体($n\geq 0$)与之联系,反之,对于实体集 B 中的每一个实体,实体集 A 中也有 m 个实体($m\geq 0$)与之联系,则称实体集 A 与

实体集 B 具有多对多联系，记为 $m:n$。

例如，一门课程同时有若干个学生选修，而一个学生可以同时选修多门课程，则课程与学生之间具有多对多联系。

实际上，一对一联系是一对多联系的特例，而一对多联系又是多对多联系的特例。

可以用图形来表示两个实体型之间的这三类联系，如图 1.9 所示。



(a) 1:1 联系　(b) 1:n 联系　(c) m:n 联系

图 1.9 两个实体型之间的三类联系

三、两个以上的实体型之间的联系

一般地，两个以上的实体型之间也存在着一对一、一对多、多对多联系。

若实体型 $E_1,E_2,\cdots,E_n$ 之间存在联系，对于实体型 $E_j(j=1,2,\cdots,i-1,i+1,\cdots,n)$ 中的给定实体，最多只和 E_i 中的一个实体相联系，则说 E_i 与 $E_1,E_2,\cdots,E_{i-1},E_{i+1},\cdots,E_n$ 之间的联系是一对多的。请读者给出多实体型之间一对一、多对多联系的定义。

例如，对于课程、教师与参考书 3 个实体型，如果一门课程可以有若干个教师讲授，使用若干本参考书，而每一个教师只讲授一门课程，每一本参考书只供一门课程使用，则课程与教师、参考书之间的联系是一对多的，如图 1.10(a)所示。

又如，有 3 个实体型：供应商、项目、零件，一个供应商可以供给多个项目多种零件，而每个项目可以使用多个供应商供应的零件，每种零件可由不同供应商供给，由此看出供应商、项目、零件三者之间是多对多的联系，如图 1.10(b)所示。要注意，3 个实体型之间多对多的联系和 3 个实体型两两之间的(3 个)多对多联系的语义是不同的。请读者给出供应商、项目、零件 3 个实体型两两之间的多对多联系的语义，并画出相应的示意图。

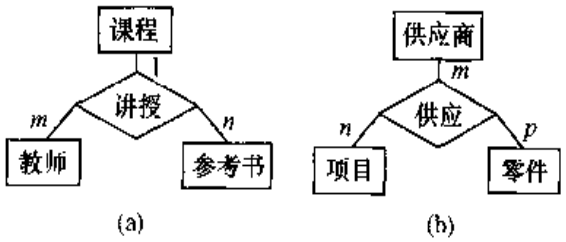


图 1.10 3 个实体型之间的联系示例

四、单个实体型内的联系

同一个实体集内的各实体之间也可以存在一对一、一对多、多对多的联系。例如，职工实体型内部具有领导与被领导的联系，即某一职工(干部)“领导”若干名职工，而一个职工

仅被另外一个职工直接领导，因此这是一对多的联系，如图 1.11 所示。

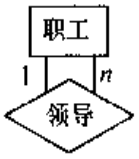


图 1.11 单个实体型之间一对多联系示例

五、概念模型的一种表示方法：实体－联系方法

概念模型是对信息世界建模，所以概念模型应该能够方便、准确地表示出上述信息世界中的常用概念。概念模型的表示方法很多，其中最为著名最为常用的是 P. P. S. Chen 于 1976 年提出的实体－联系方法(Entity-Relationship Approach)。该方法用 E-R 图来描述现实世界的概念模型，E-R 方法也称为 E-R 模型。

这里介绍 E-R 图的要点。有关如何认识和分析现实世界，从中抽取实体和实体之间的联系，建立概念模型的方法将在第七章讲解。

E-R 图提供了表示实体型、属性和联系的方法：

- 实体型：用矩形表示，矩形框内写明实体名。
- 属性：用椭圆形表示，并用无向边将其与相应的实体型连接起来。

例如，学生实体具有学号、姓名、性别、出生年份、系、入学时间等属性，用 E-R 图表示如图 1.12 所示。

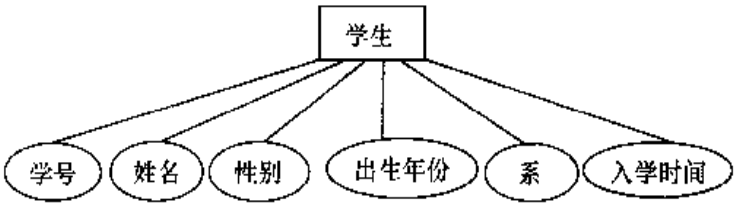


图 1.12 学生实体及属性

- 联系：用菱形表示，菱形框内写明联系名，并用无向边分别与有关实体型连接起来，同时在无向边旁标上联系的类型(1:1，1:n 或 m:n)。

需要注意的是，如果一个联系具有属性，则这些属性也要用无向边与该联系连接起来，

例如图 1.10(b)中，用“供应量”来描述联系“供应”的属性，表示某供应商供应了多少数量的零件给某个项目。那么这 3 个实体及其之间联系的 E-R 图表示可如图 1.13 所示。

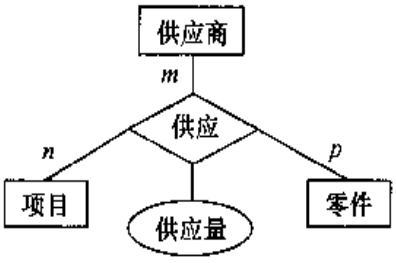


图 1.13 联系的属性

六、一个实例

下面用 E-R 图来表示某个工厂物资管理的概念模型。

物资管理涉及的实体有：

- 仓库　属性有仓库号、面积、电话号码
- 零件　属性有零件号、名称、规格、单价、描述

- 供应商 属性有供应商号、姓名、地址、电话号码、账号
- 项目 属性有项目号、预算、开工日期
- 职工 属性有职工号、姓名、年龄、职称

这些实体之间的联系如下：

(1) 一个仓库可以存放多种零件，一种零件可以存放在多个仓库中，因此仓库和零件具有多对多的联系。用库存量来表示某种零件在某个仓库中的数量。

(2) 一个仓库有多个职工当仓库保管员，一个职工只能在一个仓库工作，因此仓库和职工之间是一对多的联系。

(3) 职工之间具有领导－被领导关系。即仓库主任领导若干保管员，因此职工实体型中具有一对多的联系。

(4) 供应商、项目和零件三者之间具有多对多的联系。即一个供应商可以供给若干项目多种零件，每个项目可以使用不同供应商供应的零件，每种零件可由不同供应商供给。

下面给出此工厂的物资管理 E-R 图(如图 1.14(c)所示)。图 1.14(a)为实体及其属性图，图 1.14(b)为实体及其联系图。把实体的属性用另图画出仅仅是为了更清晰地表示实体及其实体之间的联系。

实体－联系方法是抽象和描述现实世界的有力工具。用 E-R 图表示的概念模型独立于具体的 DBMS 所支持的数据模型，它是各种数据模型的共同基础，因而比数据模型更一般、更抽象、更接近现实世界。

1.2.4 最常用的数据模型

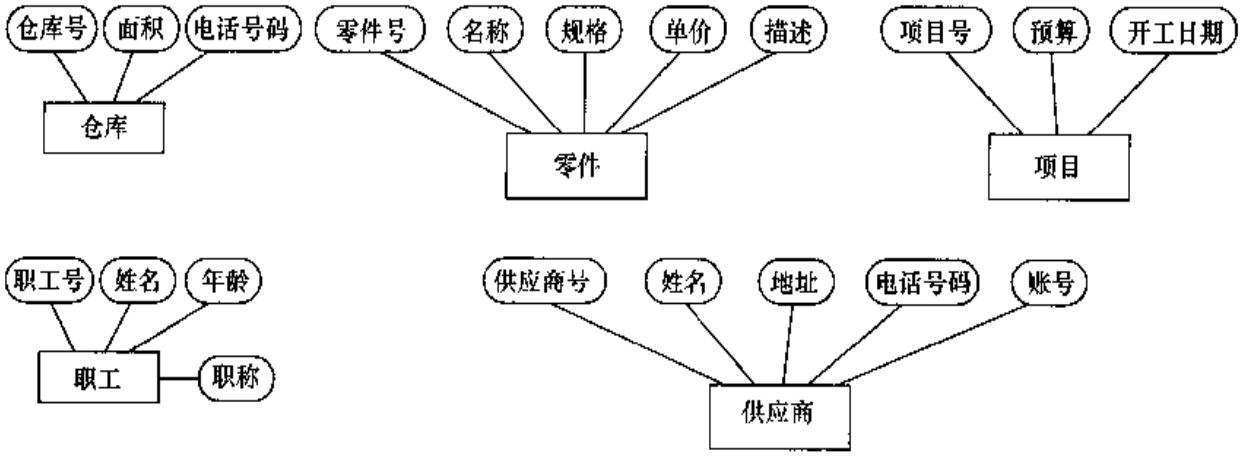
目前，数据库领域中最常用的逻辑数据模型有：

- 层次模型 (Hierarchical Model)
- 网状模型 (Network Model)
- 关系模型 (Relational Model)
- 面向对象模型 (Object Oriented Model)
- 对象关系模型 (Object Relational Model)

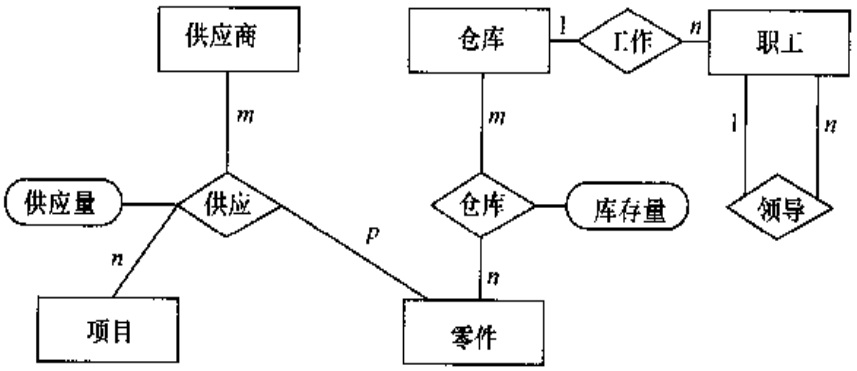
其中层次模型和网状模型统称为非关系模型。

非关系模型的数据库系统在 20 世纪 70 年代至 80 年代初非常流行，在数据库系统产品中占据了主导地位，现在已逐渐被关系模型的数据库系统取代，但在美国、欧洲等一些国家里，由于早期开发的应用系统都是基于层次数据库或网状数据库系统的，因此目前仍有不少层次数据库或网状数据库系统在继续使用。

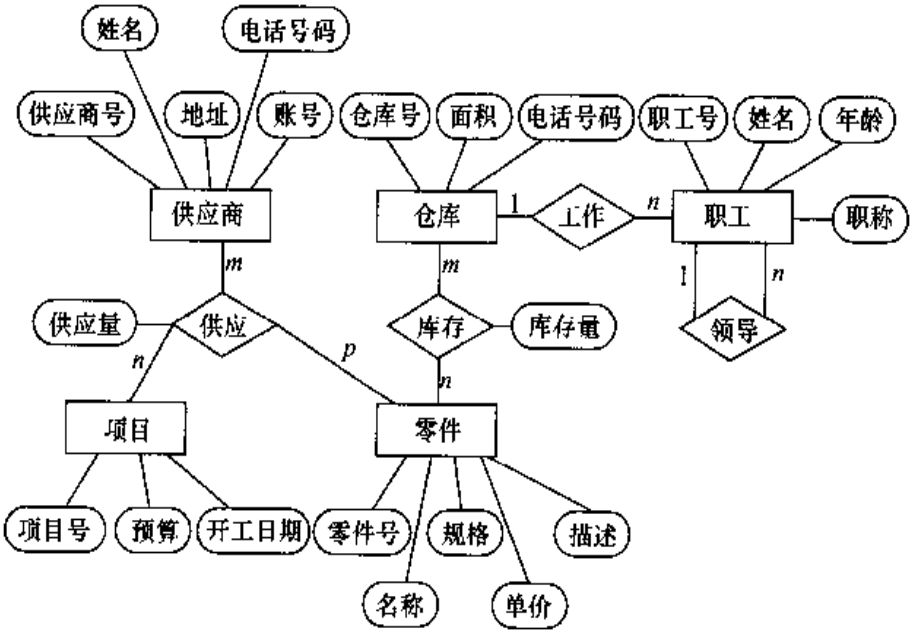
20 世纪 80 年代以来，面向对象的方法和技术在计算机各个领域，包括程序设计语言、软件工程、信息系统设计、计算机硬件设计等各方面都产生了深远的影响，也促进数据库中面向对象数据模型的研究和发展。许多关系数据库厂商为了支持面向对象模型，对关系模型



(a) 实体及其属性图



(b) 实体及其联系图



(c) 完整的实体-联系图

图 1.14 工厂物资管理 E-R 图

做了扩展，从而产生了对象关系模型。

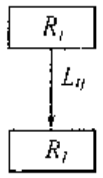
本章简要介绍层次模型、网状模型和关系模型。面向对象的数据模型将在第十五章中介绍。

数据结构、数据操作和完整性约束条件这3个方面的内容完整地描述了一个数据模型，其中数据结构是刻画模型性质的最基本的方面。为了使读者对数据模型有一个基本认识，下面着重介绍三种模型的数据结构。

注意：这里讲的数据模型都是逻辑上的，也就是说是用户眼中看到的数据范围。同时它们又都是能用某种语言描述，使计算机系统能够理解，被数据库管理系统支持的数据视图。这些数据模型将以一定的方式存储于数据库系统中，这是DBMS的功能，是DBMS中的物理存储模型。

在非关系模型中，实体用记录表示，实体的属性对应记录的数据项（或字段）。实体之间的联系在非关系模型中转换成记录之间的两两联系。

非关系模型中数据结构的单位是基本层次联系。所谓**基本层次联系**是指两个记录以及它们之间的一对多（包括一对一）的联系，如图1.15所示。



双亲结点

一对多（包括一对一）的联系名

子女结点

图1.15 基本层次联系

图中R_i位于联系L_{ij}的始点，称为**双亲结点**（Parent），R_j位于联系L_{ij}的终点，称为**子女结点**（Child）。

1.2.5 层次模型

层次模型是数据库系统中最早出现的数据模型，层次数据库系统采用层次模型作为数据的组织方式。层次数据库系统的典型代表是IBM公司的IMS（Information Management System）数据库管理系统，这是1968年IBM公司推出的第一个大型的商用数据库管理系统，曾经得到广泛的使用。

层次模型用树形结构来表示各类实体以及实体间的联系。现实世界中许多实体之间的联系本来就呈现出一种很自然的层次关系，如行政机构、家族关系等。

一、层次数据模型的数据结构

在数据库中定义满足下面两个条件的基本层次联系的集合为层次模型。

（1）有且只有一个结点没有双亲结点，这个结点称为根结点；

（2）根以外的其他结点有且只有一个双亲结点。

在层次模型中，每个结点表示一个记录类型，记录（类型）之间的联系用结点之间的连线（有向边）表示，这种联系是父子之间的一对多的联系。这就使得层次数据库系统只能处理一对多的实体联系。

每个记录类型可包含若干个字段，这里，记录类型描述的是实体，字段描述实体的属性。

各个记录类型及其字段都必须命名。各个记录类型、同一记录类型中各个字段不能同名。每个记录类型可以定义一个排序字段，也称为码字段，如果定义该排序字段的值是唯一的，则它能唯一地标识一个记录值。

一个层次模型在理论上可以包含任意有限个记录型和字段，但任何实际的系统都会因为存储容量或实现复杂度而限制层次模型中包含的记录型个数和字段的个数。

在层次模型中，同一双亲的子女结点称为兄弟结点（Twin 或 Sibling），没有子女结点的结点称为叶结点。图 1.16 给出了一个层次模型的例子。其中，R_1 为根结点；R_2 和 R_3 为兄弟结点，是 R_1 的子女结点；R_4 和 R_5 为兄弟结点，是 R_2 的子女结点；R_3、R_4 和 R_5 为叶结点。

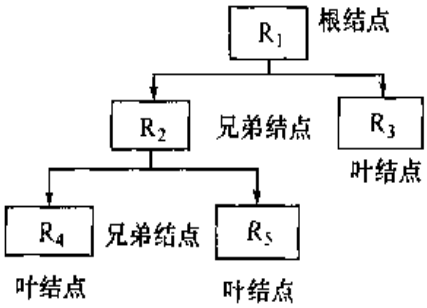


图 1.16 一个层次模型的示例

从图 1.16 上可以看出层次模型像一棵倒立的树，结点的双亲是唯一的。

层次模型的一个基本的特点是，任何一个给定的记录值只有按其路径查看时，才能显出它的全部意义，没有一个子女记录值能够脱离双亲记录值而独立存在。

图 1.17 是一个教员学生层次模型。该层次模型有 4 个记录型。记录型系是根结点，由系编号、系名、办公地点 3 个字段组成。它有两个子女结点教研室和学生。记录型教研室是系的子女结点，同时又是教员的双亲结点，它由教研室编号、教研室名两个字段组成。记录型学生由学号、姓名、成绩 3 个字段组成。记录型教员由职工号、姓名、研究方向 3 个字段组成。学生与教员是叶结点，它们没有子女结点。由系到教研室、由教研室到教员、由系到学生均是一对多的联系。

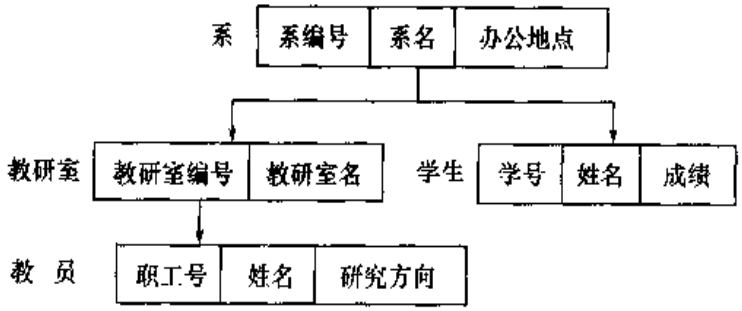


图 1.17 教员学生层次数据库模型

图 1.18 是图 1.17 数据模型对应的一个值。该值是 D02 系（计算机科学系）记录值及其所有后代记录值组成的一棵树。D02 系有 3 个教研室子女记录值：R01，R02，R03 和 3 个学生记录值：S63871，S63874，S63876。教研室 R01 有 3 个教员记录值：E2101，E1709，E3501；教研室 R03 有两个教员记录值：E1101，E3102。

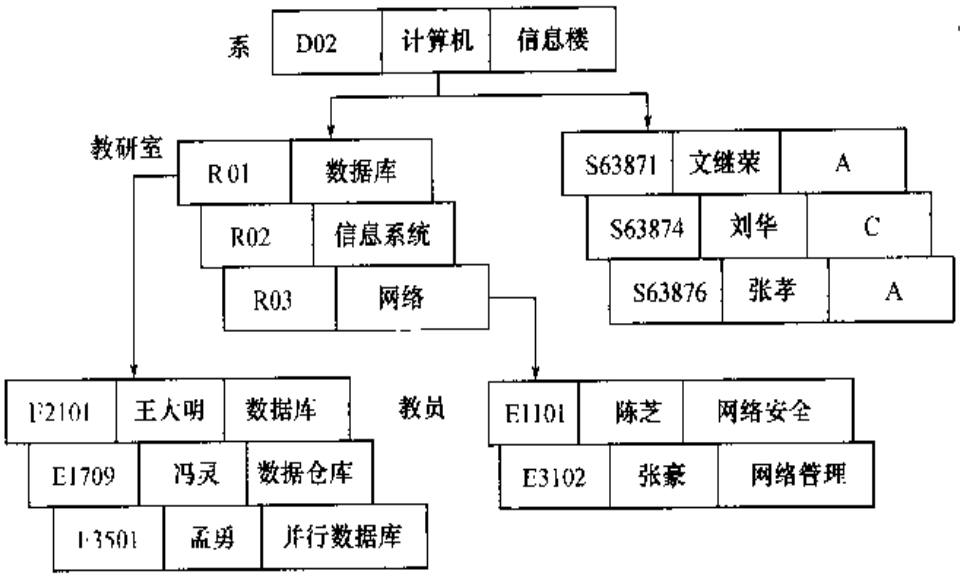


图 1.18 教员学生层次数据库的一个值

二、多对多联系在层次模型中的表示

前面已经说过,层次数据模型只能直接表示一对多(包括一对一)的联系,那么另一种常见联系,多对多联系能否在层次模型中表示呢?答案是肯定的,否则层次模型就无法真正反映现实世界了。但是用层次模型表示多对多联系,必须首先将多对多联系分解成一对多联系。分解方法有两种:冗余结点法和虚拟结点法。下面用一个例子来说明这两种分解方法。

图 1.19(a)是一个简单的多对多联系:一个学生可以选修多门课程,一门课程可由多个学生选修。学生由学号、姓名、成绩 3 个字段组成,课程由课程号和课程名两个字段组成。

图 1.19(b)采用冗余结点法,即通过增设两个冗余结点将图 1.19(a)的多对多联系转换成两个一对多联系,图 1.19(c)采用虚拟结点的分解方法,即将图 1.19(b)中的冗余结点换为虚拟结点,所谓虚拟结点就是一个指针,指向所替代的结点。

冗余结点法的优点是结构清晰,允许结点改变存储位置,缺点是需要额外占用存储空间,有潜在的不一致性。虚拟结点法的优点是减少对存储空间的浪费,避免产生潜在的不一致性,缺点是结点改变存储位置可能引起虚拟结点中指针的修改。

三、层次模型的数据操纵与完整性约束

层次模型的数据操纵主要有查询、插入、删除和更新。进行插入、删除、更新操作时要满足层次模型的完整性约束条件。

进行插入操作时,如果没有相应的双亲结点值就不能插入它的子女结点值。例如在图 1.17 的层次数据库中,若新调入一名教员,但尚未分配到某个教研室,这时就不能将新教员插入到数据库中。

进行删除操作时,如果删除双亲结点值,则相应的子女结点值也被同时删除。例如在图 1.17 的层次数据库中,若删除网络教研室,则该教研室所有教员的数据将全部丢失。

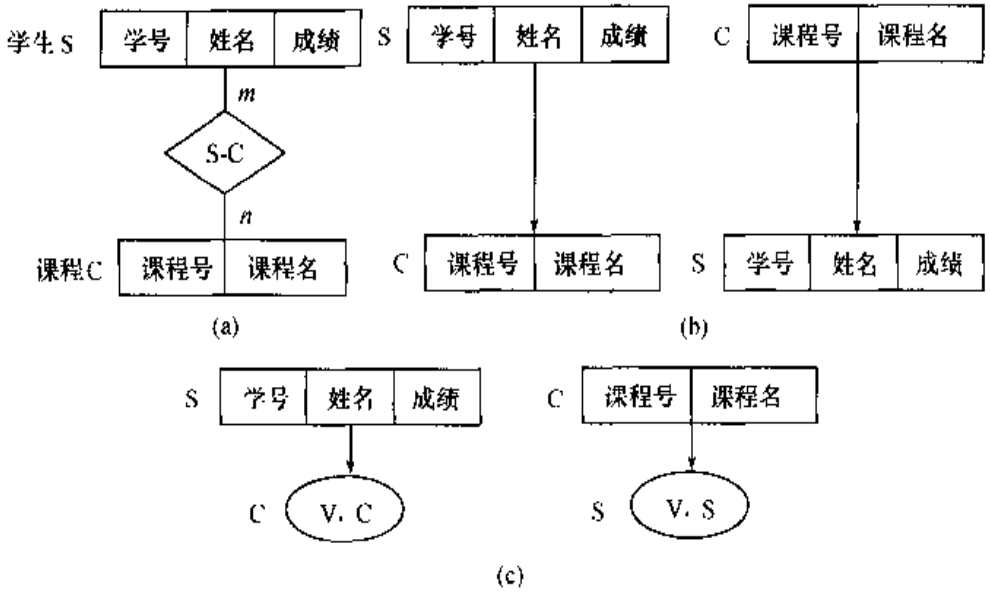


图 1.19 用层次模型表示多对多联系

四、层次数据模型的存储结构

层次数据库中不仅要存储数据本身,还要存储数据之间的层次联系。层次模型数据的存储常常是和数据之间联系的存储结合在一起的。常用的实现方法有两种:

1. 邻接法

按照层次树前序穿越的顺序把所有记录值依次邻接存放,即通过物理空间的位置相邻来体现(或隐含)层次顺序。例如对于图 1.20(a)的数据库,按邻接法存放图 1.20(b)中以根记录 A1 为首的层次记录实例集,则应如图 1.21 所示存放(为简单起见,仅用记录值的第一个字段来代表该记录值)。

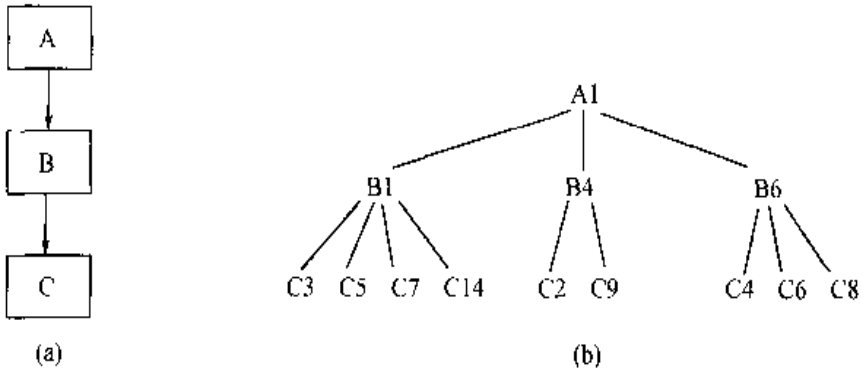


图 1.20 层次数据库及其实例

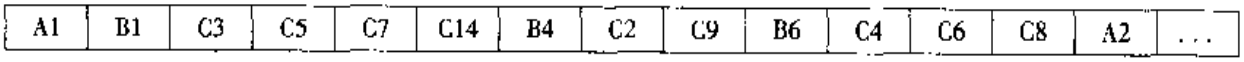
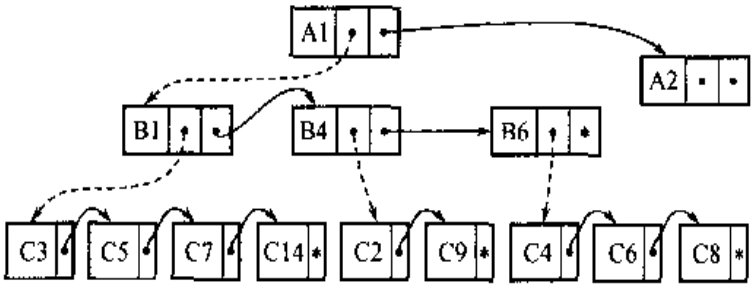


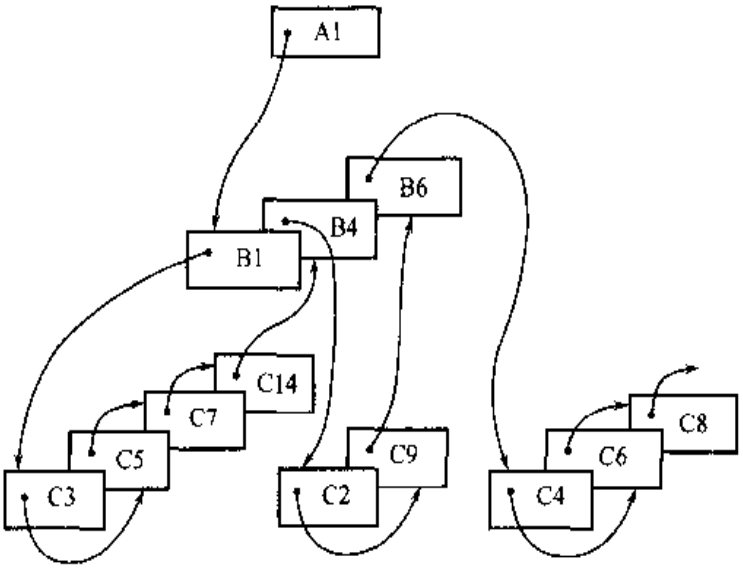
图 1.21 邻接法

2. 链接法

用指针来反映数据之间的层次联系，如图 1.22 所示。其中，图 1.22(a)中每个记录设两类指针，分别指向最左边的子女(每个记录型对应一个)和最近的兄弟，这种链接方法称为子女－兄弟链接法；图 1.22(b)是按树的前序穿越顺序链接各记录值，这种链接方法称为层次序列链接法。



(a)



(b)

图 1.22 链接法

五、层次模型的优缺点

层次模型的优点主要有：

1. 层次模型的数据结构比较简单清晰。

2. 层次数据库的查询效率高。

因为层次模型中记录之间的联系用有向边表示，这种联系在 DBMS 中常常用指针来实现。因此这种联系也就是记录之间的存取路径。当要存取某个结点的记录值，DBMS 就沿着这一条路径很快找到该记录值，所以，层次数据库的性能优于关系数据库，不低于网状数据库。

3. 层次数据模型提供了良好的完整性支持。

层次模型的缺点主要有：

1. 现实世界中很多联系是非层次性的，如结点之间具有多对多联系。

2. 一个结点具有多个双亲等，层次模型表示这类联系的方法很笨拙，只能通过引入冗余数据（易产生不一致性）或创建非自然的数据结构（引入虚拟结点）来解决。对插入和删除操作的限制比较多，因此应用程序的编写比较复杂。

3. 查询子女结点必须通过双亲结点。

4. 由于结构严密，层次命令趋于程序化。

可见用层次模型对具有一对多的层次联系的部门描述非常自然、直观，容易理解。这是层次数据库的突出优点。

1.2.6 网状模型

在现实世界中事物之间的联系更多的是非层次关系的，用层次模型表示非树形结构是很不直接的，网状模型则可以克服这一弊病。

网状数据库系统采用网状模型作为数据的组织方式。网状数据模型的典型代表是DBTG系统，亦称CODASYL系统。这是20世纪70年代数据系统语言研究会（Conference On Data System Language，CODASYL）下属的数据库任务组（Data Base Task Group，DBTG）提出的一个系统方案。DBTG系统虽然不是实际的数据库系统软件，但是它提出的基本概念、方法和技术具有普遍意义。它对于网状数据库系统的研制和发展起了重大的影响。后来不少的系统都采用DBTG模型或者简化的DBTG模型。例如，Cullinet Software公司的IDMS、Univac公司的DMS1100、Honeywell公司的IDS/2、HP公司的IMAGE等。

一、网状数据模型的数据结构

在数据库中，把满足以下两个条件的基本层次联系集合称为网状模型：

（1）允许一个以上的结点无双亲；

（2）一个结点可以有多于一个的双亲。

网状模型是一种比层次模型更具普遍性的结构。它去掉了层次模型的两个限制，允许多个结点没有双亲结点，允许结点有多个双亲结点。此外它还允许两个结点之间有多种联系（称之为复合联系）。因此，网状模型可以更直接地去描述现实世界。而层次模型实际上是网状模型的一个特例。

与层次模型一样，网状模型中每个结点表示一个记录类型（实体），每个记录类型可包含若干个字段（实体的属性），结点间的连线表示记录类型（实体）之间一对多的父子联系。

从定义可以看出，层次模型中子女结点与双亲结点的联系是唯一的，而在网状模型中这种联系可以不唯一。因此，要为每个联系命名，并指出与该联系有关的双亲记录和子女记录。例如图1.23（a）中R_3有两个双亲记录R_1和R_2，因此把R_1与R_3之间的联系命名为L_1，

R_1 与 R_3 之间的联系命名为 L_2。图 1.23(a)、(b)、(c)都是网状模型的例子。

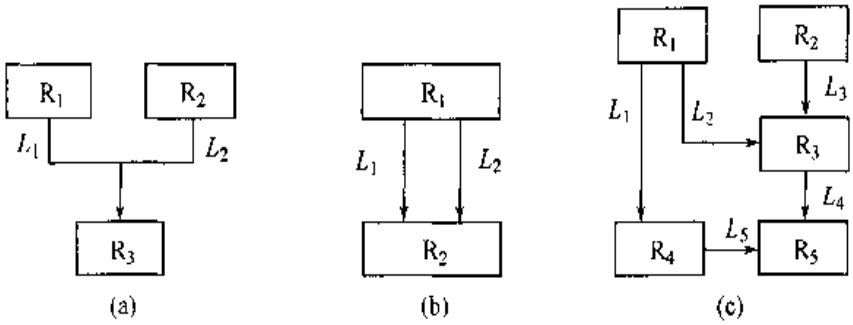


图 1.23 网状模型的例子

下面以学生选课为例,看一看网状数据库是怎样来组织数据的。

按照常规语义,一个学生可以选修若干门课程,某一课程可以被多个学生选修,因此学生与课程之间是多对多联系。这样的实体联系图不能直接用 DBTG 模型来表示。因为 DBTG 模型中不能表示记录之间多对多的联系。为此引进一个学生选课的联结记录,它由 3 个数据项组成,即学号、课程号、成绩,表示某个学生选修某一门课程及其成绩。

这样,学生选课数据库包括 3 个记录:学生、课程和选课。

每个学生可以选修多门课程,显然对学生记录中的一个值,选课记录中可以有多个值与之联系,而选课记录中的一个值,只能与学生记录中的一个值联系。学生与选课之间的联系是一对多的联系,联系名为 S-SC。同样,课程与选课之间的联系也是一对多的联系,联系名为 C-SC。图 1.24 所示为学生选课数据库的网状数据模型。

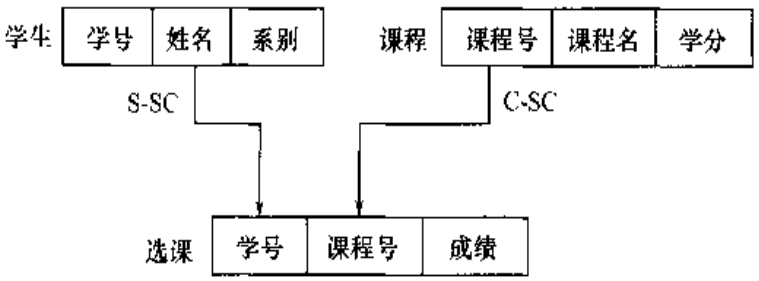


图 1.24 学生/选课/课程的网状数据模型

二、网状数据模型的操纵与完整性约束

网状数据模型一般来说没有层次模型那样严格的完整性约束条件,但具体的网状数据库系统(如 DBTG)对数据操纵都加了一些限制,提供了一定的完整性约束。

DBTG 在模式 DDL 中提供了定义 DBTG 数据库完整性的若干概念和语句,主要有:

(1) 支持记录码的概念,码即唯一标识记录的数据项的集合。例如,学生记录(如图 1.24)中学号是码,因此数据库中不允许学生记录中学号出现重复值。

(2) 保证一个联系中双亲记录和子女记录之间是一对多的联系。

(3) 可以支持双亲记录和子女记录之间某些约束条件。例如,有些子女记录要求双亲记录存在才能插入,双亲记录删除时也连同删除。例如图 1.24 中选课记录就应该满足这种约束条件,学生选课记录值必须是数据库中存在的某一学生选修存在的某一门课的记录。DBTG 提供了"属籍类别"的概念来描述这类约束条件。

三、网状数据模型的存储结构

网状数据模型的存储结构中关键是如何实现记录之间的联系。常用的方法是链接法,包括单向链接、双向链接、环状链接、向首链接等,此外还有其他实现方法,如指引元阵列法、二进制阵列法、索引法等,依具体系统不同而不同。

设图 1.24 所示学生/选课/课程网状数据库的一个实例为:

学生记录有:S1、S2、S3、S4;

课程记录有:C1、C2、C3;

S1 的选课记录有:S1C1A、S1C2A;

S2 的选课记录有:S2C1A、S2C3B;

S3 的选课记录有:S3C1B、S3C2B;

S4 的选课记录有:S4C1A、S4C2A、S4C3B;图 1.25 为学生选课网状数据库实例的一个存储示意图。

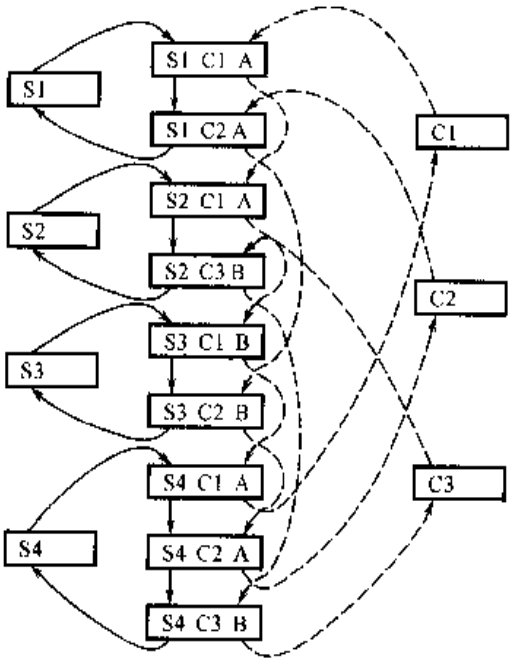


图 1.25 学生/选课/课程的网状数据库实例

学生选课数据库中学生、课程和选课 3 个记录的值可以分别按某种文件组织方式存储,记录之间的联系用单向链接法实现。

图中实线链表示 S-SC 联系,即把 S1 学生和他的选课记录(选修的 C1,C2 两门课程的选

课记录)链接起来,同样把 S2、S3、S4 学生和他们的选课记录链接起来;虚线链表示 C-SC 联系,把 C1 课程和选修了 C1 课程的学生记录(有 S1、S2、S3、S4 学生选修了 C1)链接起来,同样把 C2、C3 课程和选修了这些课程的学生记录链接起来。

四、网状数据模型的优缺点

网状数据模型的优点主要有:

1. 能够更为直接地描述现实世界,如一个结点可以有多个双亲。结点之间可以有多种联系。

2. 具有良好的性能,存取效率较高。

网状数据模型的缺点主要有:

1. 结构比较复杂,而且随着应用环境的扩大,数据库的结构就变得越来越复杂,不利于最终用户掌握。

2. 网状模型的 DDL、DML 复杂,并且要嵌入某一种高级语言(如 COBOL、C)中。用户不容易掌握,不容易使用。

由于记录之间联系是通过存取路径实现的,应用程序在访问数据时必须选择适当的存取路径,因此,用户必须了解系统结构的细节,加重了编写应用程序的负担。

1.2.7 关系模型

关系模型是目前最重要的一种数据模型。关系数据库系统采用关系模型作为数据的组织方式。

1970 年美国 IBM 公司 San Jose 研究室的研究员 E. F. Codd 首次提出了数据库系统的关系模型,开创了数据库关系方法和关系数据理论的研究,为数据库技术奠定了理论基础。由于 E. F. Codd 的杰出工作,他于 1981 年获得 ACM 图灵奖。

20 世纪 80 年代以来,计算机厂商新推出的数据库管理系统几乎都支持关系模型,非关系系统的产品也大都加上了关系接口。数据库领域当前的研究工作也都是以关系方法为基础。因此本书的重点也将放在关系数据库上,下面各章将详细讲解关系数据库。

一、关系数据模型的数据结构

关系模型与以往的模型不同,它是建立在严格的数学概念的基础上的。严格的定义将在下一章给出。这里只简单勾画一下关系模型。从用户观点看,关系模型由一组关系组成。每个关系的数据结构是一张规范化的二维表。现在以学生登记表(如图 1.26 所示)为例,介绍关系模型中的一些术语。

- 关系(Relation):一个关系对应通常说的一张表,如图 1.26 中的这张学生登记表;
- 元组(Tuple):表中的一行即为一个元组;
- 属性(Attribute):表中的一列即为一个属性,给每一个属性起一个名称即属性名。如这张表有 6 列,对应 6 个属性(学号,姓名,年龄,性别,系名和年级);

学生登记表

学号	姓名	年龄	性别	系名	年级
2005004	王小明	19	女	社会学	2005
2005006	黄大鹏	20	男	商品学	2005
2005008	张文斌	18	女	法律	2005
…	…	…	…	…	…

图 1.26 关系模型的数据结构

- 码(Key):也称为码键。表中的某个属性组,它可以唯一确定一个元组,如图 1.26 中的学号,可以唯一确定一个学生,也就成为本关系的码;
- 域(Domain):属性的取值范围,如人的年龄一般在 1 ~ 150 岁之间,大学生年龄属性的域是(14 ~ 38),性别的域是(男,女),系名的域是一个学校所有系名的集合;
- 分量:元组中的一个属性值;
- 关系模式:对关系的描述,一般表示为

 关系名(属性 1,属性 2,…,属性 n)

例如,上面的关系可描述为

学生(学号,姓名,年龄,性别,系名,年级)

在关系模型中,实体以及实体间的联系都是用关系来表示。例如,学生、课程、学生与课程之间的多对多联系在关系模型中可以如下表示:

学生(学号,姓名,年龄,性别,系名,年级)

课程(课程号,课程名,学分)

选修(学号,课程号,成绩)

关系模型要求关系必须是规范化的,即要求关系必须满足一定的规范条件,这些规范条件中最基本的一条就是,关系的每一个分量必须是一个不可分的数据项,也就是说,不允许表中还有表。图 1.27 中工资和扣除是可分的数据项,工资又分为基本工资、津贴和职务工资,扣除又分为房租和水电。因此,图 1.27 的表就不符合关系模型要求。

职工号	姓名	职称	工资			扣除		实发
			基本	津贴	职务	房租	水电	
86051	陈平	讲师	1305	1200	50	160	112	2283
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

图 1.27 一个工资表(表中有表)实例

可以把关系和现实生活中的表格所使用的术语做一个粗略的对比(表 1.2)。

表 1.2 术语对比

关系术语	一般表格的术语
关系名	表名
关系模式	表头(表格的描述)
关系	(一张)二维表
元组	记录或行
属性	列
属性名	列名
属性值	列值
分量	一条记录中的一个列值
非规范关系	表中有表(大表中嵌有小表)

二、关系数据模型的操纵与完整性约束

关系数据模型的操作主要包括查询、插入、删除和更新数据。这些操作必须满足关系的完整性约束条件。关系的完整性约束条件包括三大类:实体完整性、参照完整性和用户定义的完整性。其具体含义将在后面介绍。

关系模型中的数据操作是集合操作,操作对象和操作结果都是关系,即若干元组的集合,而不像非关系模型中那样是单记录的操作方式。另一方面,关系模型把存取路径向用户隐蔽起来,用户只要指出"干什么"或"找什么",不必详细说明"怎么干"或"怎么找",从而大大地提高了数据的独立性,提高了用户生产率。

三、关系数据模型的存储结构

在关系数据模型中,实体及实体间的联系都用表来表示。在关系数据库的物理组织中,有的 DBMS 一个表对应一个操作系统文件,有的 DBMS 从操作系统获得若干大的文件,自己设计表、索引等存储结构。

四、关系数据模型的优缺点

关系数据模型具有下列优点:

1. 关系模型与非关系模型不同,它是建立在严格的数学概念的基础上的。

2. 关系模型的概念单一。无论实体还是实体之间的联系都用关系来表示。对数据的检索和更新结果也是关系(即表)。所以其数据结构简单、清晰,用户易懂易用。

3. 关系模型的存取路径对用户透明,从而具有更高的数据独立性、更好的安全保密性,也简化了程序员的工作和数据库开发建立的工作。

所以关系数据模型诞生以后发展迅速,深受用户的喜爱。

当然,关系数据模型也有缺点,其中最主要的缺点是,由于存取路径对用户透明,查询效率往往不如非关系数据模型。因此为了提高性能,DBMS 必须对用户的查询请求进行优化,因此增加了开发 DBMS 的难度。不过,用户不必考虑这些系统内部的优化技术细节。

1.3 数据库系统结构

考察数据库系统的结构可以有多种不同的层次或不同的角度。

从数据库管理系统角度看,数据库系统通常采用三级模式结构;这是数据库管理系统内部的系统结构。

从数据库最终用户角度看,数据库系统的结构分为单用户结构、主从式结构、分布式结构、客户/服务器、浏览器/应用服务器/数据库服务器多层结构等。这是数据库系统外部的体系结构。

本章介绍数据库系统的模式结构。

1.3.1 数据库系统模式的概念

在数据模型中有“型”(Type)和“值”(Value)的概念。型是指对某一类数据的结构和属性的说明,值是型的一个具体赋值。例如,学生记录定义为(学号,姓名,性别,系别,年龄,籍贯)这样的记录型,而(900201,李明,男,计算机,22,江苏)则是该记录型的一个记录值。

模式(Schema)是数据库中全体数据的逻辑结构和特征的描述,它仅仅涉及到型的描述,不涉及到具体的值。模式的一个具体值称为模式的一个实例(Instance)。同一个模式可以有很多实例。

例如:在学生选课数据库模式中,包含学生记录、课程记录和学生选课记录,则2003年有一个学生数据库的实例,该实例包含了2003年学校中所有学生的记录(如果某校有10 000个学生,则有10 000个学生记录)、学校开设的所有课程的记录和所有学生选课的记录。

2002年度学生数据库模式对应的实例与2003年度学生数据库模式对应的实例是不同的。实际上2003年度学生数据库的实例也会随时间变化的,因为在此年度有的学生可能退学,有的学生可能转系。各个时刻学生选课数据库的实例是不同的,在变化的,而不变的是学生选课数据库模式。

模式是相对稳定的,而实例是相对变动的,因为数据库中的数据是在不断更新的。模式反映的是数据的结构及其联系,而实例反映的是数据库某一时刻的状态。

虽然实际的数据库管理系统产品种类很多,它们支持不同的数据模型,使用不同的数据库语言,建立在不同的操作系统之上,数据的存储结构也各不相同,但它们在体系结构上通常都具有相同的特征,即采用三级模式结构(早期微机上的小型数据库系统除外)并提供两级映像功能。

1.3.2 数据库系统的三级模式结构

数据库系统的三级模式结构是指数据库系统是由外模式、模式和内模式三级构成,如图

1.28 所示。

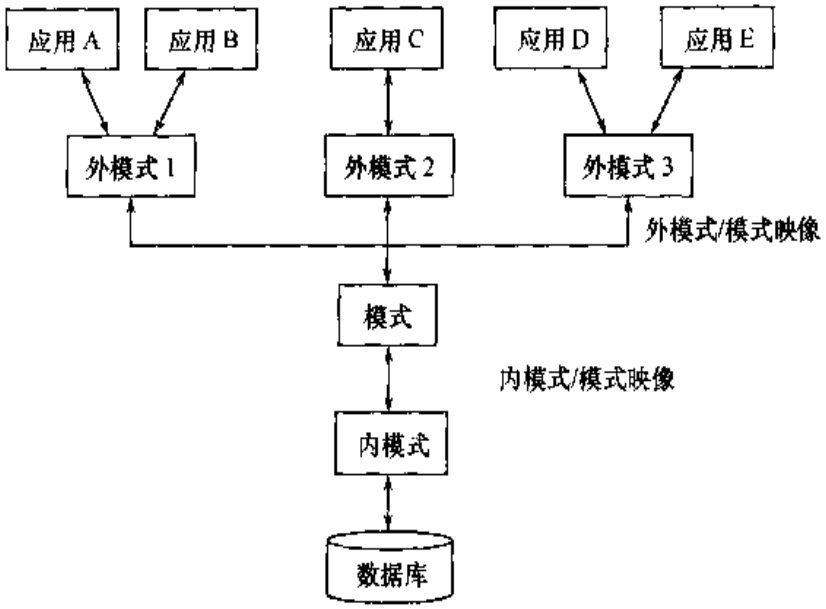


图 1.28 数据库系统的三级模式结构

一、模式(Schema)

模式也称逻辑模式,是数据库中全体数据的逻辑结构和特征的描述,是所有用户的公共数据视图。它是数据库系统模式结构的中间层,既不涉及数据的物理存储细节和硬件环境,也与具体的应用程序、所使用的应用开发工具及高级程序设计语言(如 C,COBOL,FORTRAN)无关。

模式实际上是数据库数据在逻辑级上的视图。一个数据库只有一个模式。数据库模式以某一种数据模型为基础,统一综合地考虑了所有用户的需求,并将这些需求有机地结合成一个逻辑整体。定义模式时不仅要定义数据的逻辑结构,例如数据记录由哪些数据项构成,数据项的名字、类型、取值范围等,而且要定义数据之间的联系,定义与数据有关的安全性、完整性要求。

DBSM 提供模式描述语言(模式 DDL)来严格地定义模式。

二、外模式(External Schema)

外模式也称子模式(Subschema)或用户模式,它是数据库用户(包括应用程序员和最终用户)能够看见和使用的局部数据的逻辑结构和特征的描述,是数据库用户的数据视图,是与某一应用有关的数据的逻辑表示。

外模式通常是模式的子集。一个数据库可以有多个外模式。由于它是各个用户的数据视图,如果不同的用户在应用需求、看待数据的方式、对数据保密的要求等方面存在差异,则其外模式描述就是不同的。即使对模式中同一数据,在外模式中的结构、类型、长度、保密级别等都可以不同。另一方面,同一外模式也可以为某一用户的多个应用系统所使用,但一个应用程序只能使用一个外模式。

外模式是保证数据库安全性的一个有力措施。每个用户只能看见和访问所对应的外模式中的数据,数据库中的其余数据是不可见的。

DBSM 提供子模式描述语言(子模式 DDL)来严格地定义子模式。

三、内模式(Internal Schema)

内模式也称存储模式(Storage Schema),一个数据库只有一个内模式。它是数据物理结构和存储方式的描述,是数据在数据库内部的表示方式。例如,记录的存储方式是堆存储,还是按照某个(些)属性值的升(降)序存储,还是按照属性值聚簇(cluster)存储;索引按照什么方式组织,是 B + 树索引,还是 hash 索引;数据是否压缩存储,是否加密;数据的存储记录结构有何规定,如定长结构或变长结构,一个记录不能跨物理页存储;等等。

例如学生记录,如果按堆存储,则插入一条新记录总是放在学生记录存储的最后,如图 1.29(a)所示;如果按学号升序存储,则插入一条记录就要找到它应在的位置插入,如图 1.29(b)所示;如果按照学生年龄聚簇存放,假如新插入的 S3 是 16 岁,则应插入的位置如图 1.29(c)所示。

DBSM 提供内模式描述语言(内模式 DDL,或者存储模式 DDL)来严格地定义内模式。

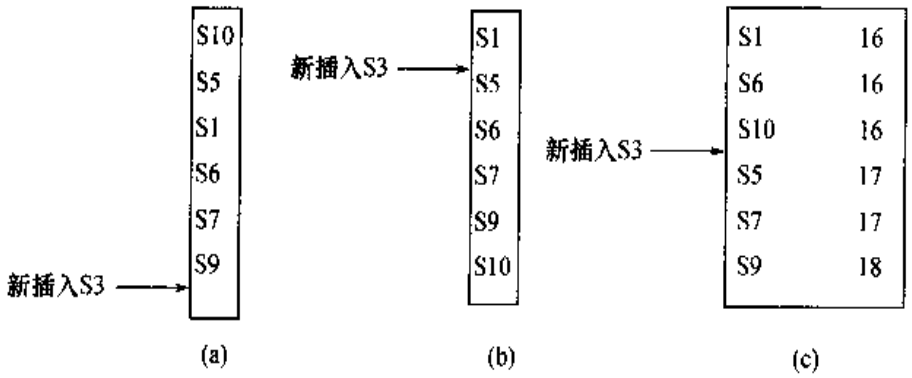


图 1.29 记录不同的存储方式示意图

1.3.3 数据库的二级映像功能与数据独立性

数据库系统的三级模式是对数据的 3 个抽象级别,它把数据的具体组织留给 DBMS 管理,使用户能逻辑地抽象地处理数据,而不必关心数据在计算机中的具体表示方式与存储方式。为了能够在系统内部实现这 3 个抽象层次的联系和转换,数据库管理系统在这三级模式之间提供了两层映像:

- 外模式/模式映像
- 模式/内模式映像

正是这两层映像保证了数据库系统中的数据能够具有较高的逻辑独立性和物理独立性。

一、外模式/模式映像

模式描述的是数据的全局逻辑结构，外模式描述的是数据的局部逻辑结构。对应于同一个模式可以有任意多个外模式。对于每一个外模式，数据库系统都有一个外模式/模式映像，它定义了该外模式与模式之间的对应关系。这些映像定义通常包含在各自外模式的描述中。

当模式改变时（例如增加新的关系、新的属性、改变属性的数据类型等），由数据库管理员对各个外模式/模式的映像作相应改变，可以使外模式保持不变。应用程序是依据数据的外模式编写的，从而应用程序不必修改，保证了数据与程序的逻辑独立性，简称数据的逻辑独立性。

二、模式/内模式映像

数据库中只有一个模式，也只有一个内模式，所以模式/内模式映像是唯一的，它定义了数据全局逻辑结构与存储结构之间的对应关系。例如，说明逻辑记录和字段在内部是如何表示的。该映像定义通常包含在模式描述中。当数据库的存储结构改变了（例如选用了另一种存储结构），由数据库管理员对模式/内模式映像作相应改变，可以使模式保持不变，从而应用程序也不必改变。保证了数据与程序的物理独立性，简称数据的物理独立性。

在数据库的三级模式结构中，数据库模式即全局逻辑结构是数据库的中心与关键，它独立于数据库的其他层次。因此设计数据库模式结构时应首先确定数据库的逻辑模式。

数据库的内模式依赖于它的全局逻辑结构，但独立于数据库的用户视图，即外模式，也独立于具体的存储设备。它是将全局逻辑结构中所定义的数据结构及其联系按照一定的物理存储策略进行组织，以达到较好的时间与空间效率。

数据库的外模式面向具体的应用程序，它定义在逻辑模式之上，但独立于存储模式和存储设备。当应用需求发生较大变化，相应外模式不能满足其视图要求时，该外模式就得做相应改动，所以设计外模式时应充分考虑到应用的扩充性。

特定的应用程序是在外模式描述的数据结构上编制的，它依赖于特定的外模式，与数据库的模式和存储结构独立。不同的应用程序有时可以共用同一个外模式。数据库的二级映像保证了数据库外模式的稳定性，从而从底层保证了应用程序的稳定性，除非应用需求本身发生变化，否则应用程序一般不需要修改。

数据与程序之间的独立性，使得数据的定义和描述可以从应用程序中分离出去。另外，由于数据的存取由 DBMS 管理，用户不必考虑存取路径等细节，从而简化了应用程序的编制，大大减少了应用程序的维护和修改。

1.4 数据库系统的组成

在本章一开始介绍了数据库系统一般由数据库、数据库管理系统（及其开发工具）、应用

系统和数据库管理员构成。下面分别介绍这几个部分的内容。

一、硬件平台及数据库

由于数据库系统数据量都很大，加之 DBMS 丰富的功能使得自身的规模也很大，因此整个数据库系统对硬件资源提出了较高的要求，这些要求是：

(1) 要有足够大的内存，存放操作系统、DBMS 的核心模块、数据缓冲区和应用程序；

(2) 有足够的大的磁盘或磁盘阵列等设备存放数据库，有足够的磁带(或光盘)作数据备份；

(3) 要求系统有较高的通道能力，以提高数据传送率。

二、软件

数据库系统的软件主要包括：

(1) DBMS。DBMS 是为数据库的建立、使用和维护配置的系统软件。

(2) 支持 DBMS 运行的操作系统。

(3) 具有与数据库接口的高级语言及其编译系统，便于开发应用程序。

(4) 以 DBMS 为核心的应用开发工具。

应用开发工具是系统为应用开发人员和最终用户提供的高效率、多功能的应用生成器、第四代语言等各种软件工具。它们为数据库系统的开发和应用提供了良好的环境。

(5) 为特定应用环境开发的数据库应用系统。

三、人员

开发、管理和使用数据库系统的人员主要是：数据库管理员、系统分析员和数据库设计人员、应用程序员和最终用户。不同的人员涉及不同的数据抽象级别，具有不同的数据视图，如图 1.30 所示，其各自的职责分别是：

1. 数据库管理员(DataBase Administrator，DBA)

在数据库系统环境下，有两类共享资源。一类是数据库，另一类是数据库管理系统软件。因此需要有专门的管理机构来监督和管理数据库系统。DBA 则是这个机构的一个(组)人员，负责全面管理和控制数据库系统。具体职责包括：

(1) 决定数据库中的信息内容和结构

数据库中要存放哪些信息，DBA 要参与决策。因此 DBA 必须参加数据库设计的全过程，并与用户、应用程序员、系统分析员密切合作共同协商，搞好数据库设计。

(2) 决定数据库的存储结构和存取策略

DBA 要综合各用户的应用要求，和数据库

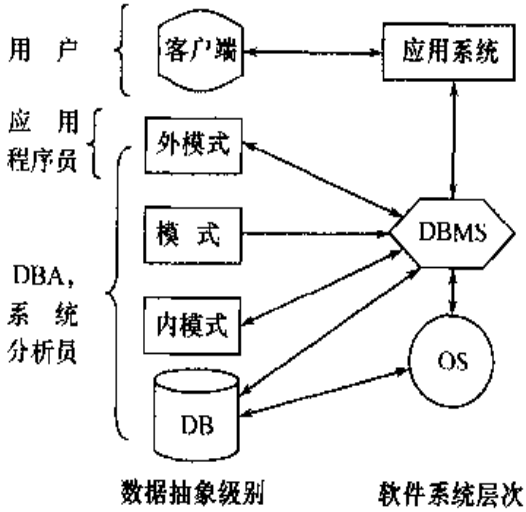


图 1.30 各种人员的数据视图

设计人员共同决定数据的存储结构和存取策略，以求获得较高的存取效率和存储空间利用率。

(3) 定义数据的安全性要求和完整性约束条件

DBA 的重要职责是保证数据库的安全性和完整性。因此 DBA 负责确定各个用户对数据库的存取权限、数据的保密级别和完整性约束条件。

(4) 监控数据库的使用和运行

DBA 还有一个重要职责就是监视数据库系统的运行情况，及时处理运行过程中出现的问题。比如系统发生各种故障时，数据库会因此遭到不同程度的破坏，DBA 必须在最短时间内将数据库恢复到正确状态，并尽可能不影响或少影响计算机系统其他部分的正常运行。为此，DBA 要定义和实施适当的后备和恢复策略。如周期性的转储数据、维护日志文件等。有关这方面的内容将在下面做进一步讨论。

(5) 数据库的改进和重组重构

DBA 还负责在系统运行期间监视系统的空间利用率、处理效率等性能指标，对运行情况进行记录、统计分析，依靠工作实践并根据实际应用环境，不断改进数据库设计。不少数据库产品都提供了对数据库运行状况进行监视和分析的工具，DBA 可以使用这些软件完成这项工作。

另外，在数据运行过程中，大量数据不断插入、删除、修改，时间一长，会影响系统的性能。因此，DBA 要定期对数据库进行重组织，以提高系统的性能。

当用户的需求增加和改变时，DBA 还要对数据库进行较大的改造，包括修改部分设计，即数据库的重构造。

2. 系统分析员和数据库设计人员

系统分析员负责应用系统的需求分析和规范说明，要和用户及 DBA 相结合，确定系统的硬件软件配置，并参与数据库系统的概要设计。

数据库设计人员负责数据库中数据的确定、数据库各级模式的设计。数据库设计人员必须参加用户需求调查和系统分析，然后进行数据库设计。在很多情况下，数据库设计人员就由数据库管理员担任。

3. 应用程序员

应用程序员负责设计和编写应用系统的程序模块，并进行调试和安装。

4. 用户

这里用户是指最终用户(End User)。最终用户通过应用系统的用户接口使用数据库。常用的接口方式有浏览器、菜单驱动、表格操作、图形显示、报表书写等。

最终用户可以分为如下三类：

(1) 偶然用户。这类用户不经常访问数据库，但每次访问数据库时往往需要不同的数据库信息，这类用户一般是企业或组织机构的高中级管理人员。

（2）简单用户。数据库的多数最终用户都是简单用户。其主要工作是查询和更新数据库，一般都是通过应用程序员精心设计并具有友好界面的应用程序存取数据库。银行的职员、航空公司的机票预定工作人员、旅馆总台服务员等都属于这类用户。

（3）复杂用户。复杂用户包括工程师、科学家、经济学家、科学技术工作者等具有较高科学技术背景的人员。这类用户一般都比较熟悉数据库管理系统的各种功能，能够直接使用数据库语言访问数据库，甚至能够基于数据库管理系统的 API 编制自己的应用程序。

1.5 小　结

本章概述了数据库的基本概念，并通过对数据管理进展情况的介绍，阐述了数据库技术产生和发展的背景，也说明了数据库系统的优点。

数据模型是数据库系统的核心和基础。本章介绍了组成数据模型的 3 个要素、概念模型和三种主要的数据库模型。

概念模型也称信息模型，用于信息世界的建模，E-R 模型是这类模型的典型代表，E-R 方法简单、清晰，应用十分广泛。

数据模型的发展经历了格式化数据模型（包括层次模型和网状模型）、关系模型，面向对象模型、对象关系模型等非传统数据模型阶段。由于层次数据库和网状数据库已逐渐被关系数据库代替，本书不再用单独的章节讲解，因此本章较为详细地讲解了层次模型和网状模型，而关系模型只是简单介绍，后面会详细讲解。

数据库系统三级模式和两层映像的系统结构保证了数据库系统中能够具有较高的逻辑独立性和物理独立性。

最后介绍了数据库系统的组成，使读者了解数据库系统不仅是一个计算机系统，而是一个人－机系统，人的作用特别是 DBA 的作用尤为重要。

学习这一章应把注意力放在掌握基本概念和基本知识方面，为进一步学习下面章节打好基础。本章新概念较多，如果是刚开始学习数据库，可在学习后面章节后再回来理解和掌握这些概念。

习　题

1. 试述数据、数据库、数据库管理系统、数据库系统的概念。
2. 使用数据库系统有什么好处？
3. 试述文件系统与数据库系统的区别和联系。
4. 举出适合用文件系统而不是数据库系统的应用例子；再举出适合用数据库系统的应用例子。
5. 试述数据库系统的特点。
6. 数据库管理系统的主要功能有哪些？

7. 试述数据模型的概念、数据模型的作用和数据模型的3个要素。

8. 试述概念模型的作用。

9. 定义并解释概念模型中以下术语：

实体，实体型，实体集，属性，码，实体－联系图(E-R图)

10. 试给出三个实际情况的E-R图，要求实体型之间具有一对一、一对多、多对多各种不同的联系。

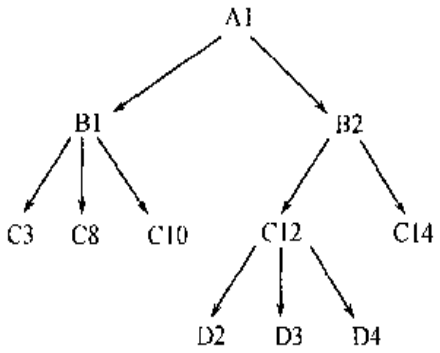
11. 试给出一个实际情况的E-R图，要求有3个实体型，而且3个实体型之间有多对多联系。3个实体型之间的多对多联系和3个实体型两两之间的3个多对多联系等价吗？为什么？

12. 学校中有若干系，每个系有若干班级和教研室，每个教研室有若干教员，其中有的教授和副教授每人各带若干研究生，每个班有若干学生，每个学生选修若干课程，每门课可由若干学生选修。请用E-R图画出此学校的概念模型。

13. 某工厂生产若干产品，每种产品由不同的零件组成，有的零件可用在不同的产品上。这些零件由不同的原材料制成，不同零件所用的材料可以相同。这些零件按所属的不同产品分别放在仓库中，原材料按照类别放在若干仓库中。请用E-R图画出此工厂产品、零件、材料、仓库的概念模型。

14. 试述层次模型的概念，举出3个层次模型的实例。

15. 今有一个层次数据库实例，试用子女－兄弟链接法和层次序列链接法画出它的存储结构示意图。



16. 试述网状模型的概念，举出3个网状模型的实例。

17. 试述网状、层次数据库的优缺点。

18. 试述关系模型的概念，定义并解释以下术语：

(1)关系　(2)属性　(3)域　(4)元组

(5)码　(6)分量　(7)关系模式

19. 试述关系数据库的特点。

20. 试述数据库系统三级模式结构，这种结构的优点是什么？

21. 定义并解释以下术语：

模式、外模式、内模式、DDL、DML

22. 什么叫数据与程序的物理独立性？什么叫数据与程序的逻辑独立性？为什么数据库系统具有数据与程序的独立性？

23. 试述数据库系统的组成。

24. DBA的职责是什么？

25. 系统分析员、数据库设计人员、应用程序员的职责是什么？

本章参考文献

1 Stonebraker M R. A Functional View of Data Independence. Proceedings of ACM SIGMOD Workshop on Data Description, Access and Control, 1974

2 Michael L. Brodie and Joachim W. Schmidt(eds.). Final Report of the ANSI/X3/SPARC DBS-SG Relational Database Task Group, ACM SIGMOD Record 12, No. 4, 1982

3 ANSI/X3/SPARC Study Group on Data Base Management Systems. Interim Report, FDT(ACM SIGMOD bulletin)7. No. 2, 1975

4 Information on IMS is available from IBM Corporation, Armonk, New York 10504

(文献[4],[5]全面介绍了 IMS。)

5 McGee W C. The Information Management System IMS/VS. Part I: General Structure and Operation. IBM Systems Journal, 6:2, 1977

6 Kaoo D, Leben J F. IMS Programming Techniques: A Guide to Using DL/I. New York, Van Nostrand Reinhold, 1978

(本文全面介绍了 IMS 系统的语言 DL/I。)

7 Bjorner D, Lovengren H. Formalization of Database Systems and a Formal Definition of IMS. in Proceedings of VLDB, 1982

(本文把 IMS 的数据模型进行了形式化,给出了比较严格的层次数据模型的定义。)

8 Tsichritzis D C, Lochovsky F H. Hierarchical Data Base Management: A Survey. ACM Computing Surveys, 8:1, 1976

(本文介绍了层次数据库系统的一般原理。)

9 萨师煊,王珊. 数据库系统概论(第二版). 北京:高等教育出版社,1991

(本书第二、第三章介绍了层次数据库系统和网状数据库系统的基本概念和一般原理。)

10 Bachman C, Williams S. A General Purpose Programming System for Random Access Memories. Proceedings of the Fall Joint Computer Conference, AFIPS, 26, 19641. 11

(本文献描述的是 Charles Bachman 在第一个商业 DBMS 的开发期间进行的网状数据模型早期研究工作。)

11 Bachman C. Data Structure Diagrams. Data Base (Bulletin of ACM SIGFIDET), 1:2, 1969

12 Bachman C. The Programmer as a Navigator. CACM, 16:1, 1973

(本文献最早提出了用数据结构简图表示数据库之间联系的思想。)

13 Bachman C. The Data Structure Set Model. In Proceedings of the ACM SIGMOD Debate on Data Model: Data Structure Set Versus Relation, 1974

(Bachman 的成就使他于 1973 年荣获了 ACM 最高荣誉——图灵奖。他在图灵奖演说中把数据库看作是基本资源,把程序设计者看作是数据库中的领航员。)

14 A Survey of Generalized Data Base Management Systems. CODASYL Systems Committee Technical Report (ACM and IAG.), 1969

(在 1974 年支持和反对关系模型研究的讨论中, Bachman 持反对态度。)

15 Data Base Task Group of CODASYL Programming Language Committee Report,1971

16 Schenk H. Implementation Aspects of the DBTG Proposal. Proceedings of the IFIP Working Conference on Database Management Systems,1974

(CODASYL 的数据库任务组(DBTG)对网状数据库系统进行了系统研究,在 1971 年的报告[15]中提出了模式 DDL、子模式 DDL 和嵌入 COBOL 的 DML,以后又发表了多篇报告。)

17 Taylor R,Frank R. CODASYL Data Base Management Systems. ACM Computing Surveys,8:1,1976

18 CODASYL Data Description Language Committee Journal of Development,Canadian Government Publishing Center,1978

(本文对 CODASYL 网状数据库管理进行了综述。)

19 Dahl R,Bubenko J. IDBD:An Interactive Design Tool for CODASYL DBTG Type Databases. In Proceedings of VLDB,1982

(本文讨论了网状数据库语言。)

20 Cardenas A. Data Base Management Systems(Second Edition). Allyn and Bacon,1985

(本文讨论了网状数据库设计。)

21 Olle T. The CODASYL Approach to Data Base Management. Wiley,1978

22 Griethuysen J J Van (ed.),Concepts and Terminology for the Conceptual Schema and the Information Base. International Standards Organization Document No. ISO/TC97/SC5N695,1982

23 Jay-Louise Weldon. Data Base Administration. New York,N Y:Plenum Press,1981

第二章 关系数据库

关系数据库应用数学方法来处理数据库中的数据。最早将这类方法用于数据处理的是1962年CODASYL发表的“信息代数”,之后有1968年David Child在IBM 7090机上实现的集合论数据结构,但系统地、严格地提出关系模型的是美国IBM公司的E. F. Codd。

1970年E. F. Codd在美国计算机学会会刊《Communications of the ACM》上发表的题为“A Relational Model of Data for Shared Data Banks”的论文,开创了数据库系统的新纪元。ACM在1983年把这篇论文列为从1958年以来的四分之一世纪中具有里程碑意义的25篇研究论文之一。以后,他连续发表了多篇论文,奠定了关系数据库的理论基础。

20世纪70年代末,关系方法的理论研究和软件系统的研制均取得了很大成果,IBM公司的San Jose实验室在IBM370系列机上研制的关系数据库实验系统System R历时6年获得成功。1981年IBM公司又宣布了具有System R全部特征的新的数据库软件产品SQL/DS问世。

与System R同期,美国加州大学伯克利分校也研制了INGRES关系数据库实验系统,并由INGRES公司发展成为INGRES数据库产品。

30多年来,关系数据库系统的研究和开发取得了辉煌的成就。关系数据库系统从实验室走向了社会,成为最重要、应用最广泛的数据库系统,大大促进了数据库应用领域的扩大和深入。因此,关系数据模型的原理、技术和应用十分重要,是本书、本课程的重点。

本书第二~六章将集中讨论关系数据库的有关问题。其中:第二章讲解关系模型的基本概念,即关系模型的数据结构、关系操作和关系的完整性;第三、四、五章介绍关系数据库标准语言SQL的数据定义、数据查询、数据更新、数据控制等功能;第六章介绍关系数据理论,这是关系数据库的理论基础,也是关系数据库系统逻辑设计的工具。

2.1 关系数据结构及形式化定义

关系数据库系统是支持关系模型的数据库系统。第1章初步介绍了关系模型,介绍了关系模型的一些基本术语。这一章将较深入地讲解关系模型。

按照数据模型的3个要素,关系模型由关系数据结构、关系操作集合和关系完整性约束3部分组成。下面将分别介绍关系模型的3个方面。其中2.1节讲解关系数据结构,包括关系的形式化定义及有关概念;2.2节讲解关系操作;2.3节讲解关系的三类完整性约束;2.4节中讲解关系代数。这是关系数据库系统中实现关系操作的一种语言。最后,在2.5节介绍关系演算。

2.1.1 关系

关系模型的数据结构非常简单,只包含单一的数据结构——关系。

在用户看来,关系模型中数据的逻辑结构是一张扁平的二维表。

关系模型的数据结构虽然简单却能够表达丰富的语义,描述出现实世界的实体以及实体间的各种联系。也就是说,在关系模型中,现实世界的实体以及实体间的各种联系均用单一的结构类型即关系来表示。

前面已经非形式化地介绍了关系模型及有关的基本概念。关系模型是建立在集合代数的基础上的,这里从集合论角度给出关系数据结构的形式化定义。

1. 域(Domain)

定义 2.1 域是一组具有相同数据类型的值的集合。

例如,自然数、整数、实数、长度小于25字节的字符串集合、{0,1}、大于等于0且小于等于100的正整数等,都可以是域。

2. 笛卡尔积(Cartesian Product)

定义 2.2 给定一组域 $D_1, D_2, \cdots, D_n$,这些域中可以是相同的域。$D_1, D_2, \cdots, D_n$ 的**笛卡尔积**为

$$D_1 \times D_2 \times \cdots \times D_n = \{(d_1, d_2, \cdots, d_n) \mid d_i \in D_i, i = 1, 2, \cdots, n\}$$

其中每一个元素 $(d_1, d_2, \cdots, d_n)$ 叫作一个 ***n* 元组**(n-tuple)或简称**元组**(Tuple)。

元素中的每一个值 d_i 叫作一个**分量**(Component)。

这些域中可以存在相同的域。例如 D_2 和 D_3 可以是相同的域。

若 $D_i (i = 1, 2, \cdots, n)$ 为有限集,其基数(Cardinal number)为 $m_i (i = 1, 2, \cdots, n)$,则 $D_1 \times D_2 \times \cdots \times D_n$ 的基数 M 为:

$$M = \prod_{i=1}^{n} m_i$$

笛卡尔积可表示为一个二维表。表中的每行对应一个元组,表中的每一列的值来自一个域。例如给出3个域:

D_1 = 导师集合 SUPERVISOR = 张清玫,刘逸

D_2 = 专业集合 SPECIALITY = 计算机专业,信息专业

D_3 = 研究生集合 POSTGRADUATE = 李勇,刘晨,王敏

则 D_1, D_2, D_3 的笛卡尔积为

$D_1 \times D_2 \times D_3 =$ {
(张清玫,计算机专业,李勇),(张清玫,计算机专业,刘晨),
(张清玫,计算机专业,王敏),(张清玫,信息专业,李勇),
(张清玫,信息专业,刘晨),(张清玫,信息专业,王敏),
(刘逸,计算机专业,李勇),(刘逸,计算机专业,刘晨),
(刘逸,计算机专业,王敏),(刘逸,信息专业,李勇),
(刘逸,信息专业,刘晨),(刘逸,信息专业,王敏) }

其中(张清玫,计算机专业,李勇)、(张清玫,计算机专业,刘晨)等都是元组。张清玫、计算机专业、李勇、刘晨等都是分量。

该笛卡尔积的基数为 $2 \times 2 \times 3 = 12$,也就是说,$D_1 \times D_2 \times D_3$ 一共有 $2 \times 2 \times 3 = 12$ 个元组。这 12 个元组可列成一张二维表(如表 2.1)。

表 2.1 D_1、D_2、D_3 的笛卡尔积

SUPERVISOR	SPECIALITY	POSTGRADUATE
张清玫	计算机专业	李勇
张清玫	计算机专业	刘晨
张清玫	计算机专业	王敏
张清玫	信息专业	李勇
张清玫	信息专业	刘晨
张清玫	信息专业	王敏
刘逸	计算机专业	李勇
刘逸	计算机专业	刘晨
刘逸	计算机专业	王敏
刘逸	信息专业	李勇
刘逸	信息专业	刘晨
刘逸	信息专业	王敏

3. 关系(Relation)

定义 2.3 $D_1 \times D_2 \times \cdots \times D_n$ 的子集叫做在域 $D_1, D_2, \cdots, D_n$ 上的**关系**,表示为

$$R(D_1, D_2, \cdots, D_n)$$

这里 R 表示关系的名字,n 是关系的**目或度**(Degree)。

关系中的每个元素是关系中的元组,通常用 t 表示。

当 $n = 1$ 时,称该关系为**单元关系**(Unary relation)。

当 $n = 2$ 时,称该关系为**二元关系**(Binary relation)。

关系是笛卡尔积的有限子集,所以关系也是一个二维表,表的每行对应一个元组,表的

每列对应一个域。由于域可以相同，为了加以区分，必须对每列起一个名字，称为属性(Attribute)。n 目关系必有 n 个属性。

若关系中的某一属性组的值能唯一地标识一个元组，则称该属性组为**候选码**(Candidate key)。

若一个关系有多个候选码，则选定其中一个为**主码**(Primary key)。

候选码的诸属性称为**主属性**(Prime attribute)。不包含在任何候选码中的属性称为**非主属性**(Nonprimary attribute)或**非码属性**(Non-key attribute)。

在最简单的情况下，候选码只包含一个属性。在最极端的情况下，关系模式的所有属性是这个关系模式的候选码，称为**全码**(All-key)。

一般来说，$D_1, D_2, \cdots, D_n$ 的笛卡尔积是没有实际语义的。只有它的某个子集才有实际含义。

例如，可以发现，表 2.1 的笛卡尔积中许多元组是没有意义的。因为在学校中一个专业方向有多个导师，而一个导师只在一个专业方向带研究生(专业方向与导师是一对多的联系)；一个导师可以带多名研究生，而一名研究生只有一个导师，学习某一个专业(导师与研究生是一对多的联系)。因此，表 2.1 中的一个子集才是有意义的，才可以表示导师与研究生的关系，把该关系取名为 SAP。如表 2.2 所示，李勇和刘晨是计算机专业张清玫老师的研究生；王敏是信息专业刘逸老师的研究生。

表 2.2 SAP 关系

SUPERVISOR	SPECIALITY	POSTGRADUATE
张清玫	计算机专业	李勇
张清玫	计算机专业	刘晨
刘逸	信息专业	王敏

把关系 SAP 的属性名就取域名，即 SUPERVISOR，SPECIALITY 和 POSTGRADUATE。则这个关系可以表示为

SAP(SUPERVISOR, SPECIALITY, POSTGRADUATE)

假设研究生不会重名(这在实际当中是不合适的，这里只是为了举例方便)，则 POSTGRADUATE 属性的每一个值都唯一地标识了一个元组，因此可以作为 SAP 关系的主码。

关系可以有三种类型：基本关系(通常又称为基本表或基表)、查询表和视图表。

- 基本表是实际存在的表，它是实际存储数据的逻辑表示。
- 查询表是查询结果对应的表。
- 视图表是由基本表或其他视图表导出的表，是虚表，不对应实际存储的数据。

按照**定义 2.2**，关系可以是一个无限集合。由于笛卡尔积不满足交换律，所以按照数学

定义，$(d_1,d_2,\cdots,d_n)\neq(d_2,d_1,\cdots,d_n)$。当关系作为关系数据模型的数据结构时，需要给予如下的限定和扩充：

（1）无限关系在数据库系统中是无意义的。因此，限定关系数据模型中的关系必须是有限集合。

（2）通过为关系的每个列附加一个属性名的方法取消关系元组的有序性，即$(d_1,d_2,\cdots,d_i,d_j,\cdots,d_n)=(d_1,d_2,\cdots,d_j,d_i,\cdots,d_n)$ $(i,j=1,2,\cdots,n)$。

因此，基本关系具有以下6条性质：

① 列是同质的(Homogeneous)，即每一列中的分量是同一类型的数据，来自同一个域。

② 不同的列可出自同一个域，称其中的每一列为一个属性，不同的属性要给予不同的属性名。

例如，在上面的例子中，也可以只给出两个域：

人(PERSON)=张清玫，刘逸，李勇，刘晨，王敏

专业(SPECIALITY)=计算机专业，信息专业

SAP关系的导师属性和研究生属性都从PERSON域中取值。为了避免混淆，必须给这两个属性取不同的属性名，而不能直接使用域名。例如，定义导师属性名为SUPERVISOR-PERSON(或SUPERVISOR)，研究生属性名为POSTGRADUATE-PERSON(或POSTGRADUATE)。

③ 列的顺序无所谓，即列的次序可以任意交换。

由于列顺序是无关紧要的，因此在许多实际关系数据库产品中，增加新属性时，永远是插至最后一列。

④ 任意两个元组的候选码不能相同。

⑤ 行的顺序无所谓，即行的次序可以任意交换。

⑥ 分量必须取原子值，即每一个分量都必须是不可分的数据项。

关系模型要求关系必须是规范化的，即要求关系必须满足一定的规范条件。这些规范条件中最基本的一条就是，关系的每一个分量必须是一个不可分的数据项。规范化的关系简称为范式(Normal Form)。范式的概念将在第六章中做进一步讲解。

例如，表2.3虽然很好地表达了导师与研究生之间的一对多关系，但由于属性POSTGRADUATE中分量取了两个值，不符合规范化的要求，因此这样的关系在数据库中是不允许的。通俗地讲，关系表中不允许还有表，简言之不允许“表中有表”。直观地描述，表2.3中还有一个小表。

注意：在许多实际关系数据库产品中，基本表并不完全具有这6条性质，例如，有的数据库产品仍然区分了属性顺序和元组的顺序；有的关系数据库产品中允许关系表中存在两个完全相同的元组，除非用户特别定义了相应的约束条件。

表 2.3 非规范化关系

SUPERVISOR	SPECIALITY	POSTGRADUATE	
		PG1	PG2
张清玫	信息专业	李勇	刘晨
刘逸	信息专业	王敏	

小表

2.1.2 关系模式

在数据库中要区分型和值。关系数据库中，关系模式是型，关系是值。关系模式是对关系的描述，那么一个关系需要描述哪些方面呢？

首先，应该知道，关系实质上是一张二维表，表的每一行为一个元组，每一列为一个属性。一个元组就是该关系所涉及的属性集的笛卡尔积的一个元素。关系是元组的集合，因此关系模式必须指出这个元组集合的结构，即它由哪些属性构成，这些属性来自哪些域，以及属性与域之间的映像关系。

其次，一个关系通常是由赋予它的元组语义来确定的。元组语义实质上是一个 n 目谓词（n 是属性集中属性的个数）。凡使该 n 目谓词为真的笛卡尔积中的元素（或者说凡符合元组语义的那部分元素）的全体就构成了该关系模式的关系。

现实世界随着时间在不断地变化，因而在不同的时刻，关系模式的关系也会有所变化。但是，现实世界的许多已有事实限定了关系模式所有可能的关系必须满足一定的完整性约束条件。这些约束或者通过对属性取值范围的限定，例如职工年龄小于 60 岁（60 岁以后退休），或者通过属性值间的相互关连（主要体现于值的相等与否）反映出来。关系模式应当刻划出这些完整性约束条件。

定义 2.4 关系的描述称为**关系模式**（Relation Schema）。它可以形式化地表示为

$$R(U,D,\mathrm{DOM},F)$$

其中 R 为关系名，U 为组成该关系的属性名集合，D 为属性组 U 中属性所来自的域，DOM 为属性向域的映像集合，F 为属性间数据的依赖关系集合。

属性间的数据依赖将在第六章讨论，本章中关系模式仅涉及关系名、各属性名、域名、属性向域的映像四部分，即 $R(U,D,\mathrm{DOM})$。

例如，在上面例子中，由于导师和研究生出自同一个域 ——人，所以要取不同的属性名，并在模式中定义属性向域的映像，即说明它们分别出自哪个域，如：

$$\mathrm{dom}(\mathrm{SUPERVISOR\text{-}PERSON}) = \mathrm{dom}(\mathrm{POSTGRADUATE\text{-}PERSON}) = \mathrm{PERSON}$$

关系模式通常可以简记为

$$R(U)$$

或

$$R(A_1,A_2,\cdots,A_n)$$

其中 R 为关系名，$A_1, A_2, \cdots, A_n$ 为属性名。而域名及属性向域的映像常常直接说明为属性的类型、长度。

关系是关系模式在某一时刻的状态或内容。关系模式是静态的、稳定的，而关系是动态的、随时间不断变化的，因为关系操作在不断地更新着数据库中的数据。在实际工作中，人们常常把关系模式和关系都笼统地称为关系，这不难从上下文中加以区别，希望读者注意。

2.1.3 关系数据库

在关系模型中，实体以及实体间的联系都是用关系来表示的。例如导师实体、研究生实体、导师与研究生之间的一对多联系都可以分别用一个关系来表示。在一个给定的应用领域中，所有实体及实体之间联系的关系的集合构成一个关系数据库。

关系数据库也有型和值之分。关系数据库的型也称为关系数据库模式，是对关系数据库的描述。关系数据库模式包括：

- 若干域的定义；
- 在这些域上定义的若干关系模式。

关系数据库的值是这些关系模式在某一时刻对应的关系的集合，通常就称为关系数据库。

2.2 关系操作

关系模型由关系数据结构、关系操作集合和关系完整性约束三部分组成。2.1 节讲解了关系数据结构，这一节讲解关系操作的一般概念和分类。

关系模型给出了关系操作的能力的说明，但不对 RDBMS 语言给出具体的语法要求，也就是说不同的 RDBMS 可以定义和开发不同的语言来实现这些操作。

2.2.1 基本的关系操作

关系模型中常用的关系操作包括查询(Query)操作和插入(Insert)、删除(Delete)、修改(Update)操作两大部分。

关系的查询表达能力很强，是关系操作中最主要的部分。查询操作又可以分为：选择(Select)、投影(Project)、连接(Join)、除(Divide)、并(Union)、差(Except)、交(Intersection)、笛卡尔积等。

其中选择、投影、并、差、笛卡尔积是 5 种基本操作。其他操作是可以用基本操作来定义和导出的。就像乘法可以用加法来定义和导出一样。

关系操作的特点是集合操作方式，即操作的对象和结果都是集合。这种操作方式也称为一次一集合(set-at-a-time)的方式。相应地，非关系数据模型的数据操作方式则为一次一

记录(record-at-a-time)的方式。

2.2.2 关系数据语言的分类

早期的关系操作能力通常用代数方式或逻辑方式来表示,分别称为关系代数和关系演算。关系代数是用对关系的运算来表达查询要求的。关系演算是用谓词来表达查询要求的。关系演算又可按谓词变元的基本对象是元组变量还是域变量分为元组关系演算和域关系演算。关系代数、元组关系演算和域关系演算三种语言在表达能力上是完全等价的。

关系代数、元组关系演算和域关系演算均是抽象的查询语言,这些抽象的语言与具体的RDBMS中实现的实际语言并不完全一样。但它们能用作评估实际系统中查询语言能力的标准或基础。实际的查询语言除了提供关系代数或关系演算的功能外,还提供了许多附加功能,例如聚集函数(Aggregation function)、关系赋值、算术运算等等。使得目前实际查询语言功能十分强大。

另外还有一种介于关系代数和关系演算之间的语言SQL(Structured Query Language)。SQL不仅具有丰富的查询功能,而且具有数据定义和数据控制功能,是集查询、DDL、DML和DCL于一体的关系数据语言。它充分体现了关系数据语言的特点和优点,是关系数据库的标准语言。

因此,关系数据语言可以分为三类:

- 关系数据语言
 - 关系代数语言 —— 例如 ISBL
 - 关系演算语言
 - 元组关系演算语言 —— 例如 APLHA,QUEL
 - 域关系演算语言 —— 例如 QBE
 - 具有关系代数和关系演算双重特点的语言 —— 例如 SQL

这些关系数据语言的共同特点是,语言具有完备的表达能力,是非过程化的集合操作语言,功能强,能够嵌入高级语言中使用。

关系语言是一种高度非过程化的语言,用户不必请求DBA为其建立特殊的存取路径,存取路径的选择由RDBMS的优化机制来完成。例如,在一个存储有几百万条记录的关系中查找符合条件的某一个或某一些记录,从原理上讲可以有多种查找方法。例如,可以顺序扫描这个关系,也可以通过某一种索引来查找。不同的查找路径(或者称为存取路径)的效率是不同的,有的完成某一个查询可能很快,有的可能极慢。RDBMS中研究和开发了查询优化方法,系统可以自动地选择较优的存取路径,提高查询效率。

2.3 关系的完整性

关系模型的完整性规则是对关系的某种约束条件。也就是说关系的值随着时间变化时应该满足一些约束条件。这些约束条件实际上是现实世界的要求。任何关系在任何时刻都

要满足这些语义约束。

2.3.1 关系的三类完整性约束

关系模型中有三类完整性约束：实体完整性、参照完整性和用户定义的完整性。其中实体完整性和参照完整性是关系模型必须满足的完整性约束条件，被称作是关系的两个**不变性**，应该由关系系统自动支持。用户定义的完整性是应用领域需要遵循的约束条件，体现了具体领域中的语义约束。

2.3.2 实体完整性(Entity Integrity)

规则 2.1 实体完整性规则 若属性(指一个或一组属性)A 是基本关系 R 的主属性，则 A 不能取空值。

所谓空值就是“不知道”或“不存在”的值。

例如，在关系“SAP(SUPERVISOR, SPECIALITY, POSTGRADUATE)”中，研究生姓名——POSTGRADUATE 这个属性为主码(假设研究生不会重名)，则 POSTGRADUATE 不能取空值。

按照实体完整性规则的规定，基本关系的主码都不能取空值。如果主码由若干属性组成，则所有这些主属性都不能取空值。例如学生选课关系——选修(学号,课程号,成绩)中，“学号、课程号”为主码，则“学号”和“课程号”两个属性都不能取空值。

对于实体完整性规则说明如下：

(1) 实体完整性规则是针对基本关系而言的。一个基本表通常对应现实世界的一个实体集。例如学生关系对应于学生的集合。

(2) 现实世界中的实体是可区分的，即它们具有某种唯一性标识。例如每个学生都是独立的个体，是不一样的。

(3) 相应地，关系模型中以主码作为唯一性标识。

(4) 主码中的属性即主属性不能取空值。如果主属性取空值，就说明存在某个不可标识的实体，即存在不可区分的实体，这与第(2)点相矛盾，因此这个规则称为实体完整性。

2.3.3 参照完整性(Referential Integrity)

现实世界中的实体之间往往存在某种联系，在关系模型中实体及实体间的联系都是用关系来描述的。这样就自然存在着关系与关系间的引用。先来看 3 个例子。

[例 1] 学生实体和专业实体可以用下面的关系来表示，其中主码用下划线标识：

学生(学号,姓名,性别,专业号,年龄)

专业(专业号,专业名)

这两个关系之间存在着属性的引用，即学生关系引用了专业关系的主码“专业号”。显然，学生关系中的“专业号”值必须是确实存在的专业的专业号，即专业关系中有该专业的记

求。也就是说,学生关系中的某个属性的取值需要参照专业关系的属性取值。

[例 2] 学生、课程、学生与课程之间的多对多联系可以如下 3 个关系表示:

学生(学号,姓名,性别,专业号,年龄)
课程(课程号,课程名,学分)
选修(学号,课程号,成绩)

这 3 个关系之间也存在着属性的引用,即选修关系引用了学生关系的主码"学号"和课程关系的主码"课程号"。同样,选修关系中的"学号"值必须是确实存在的学生的学号,即学生关系中有该学生的记录;选修关系中的"课程号"值也必须是确实存在的课程的课程号,即课程关系中有该课程的记录。换句话说,选修关系中某些属性的取值需要参照其他关系的属性取值。

不仅两个或两个以上的关系间可以存在引用关系,同一关系内部属性间也可能存在引用关系。

[例 3] 在学生(学号,姓名,性别,专业号,年龄,班长)关系中,"学号"属性是主码,"班长"属性表示该学生所在班级的班长的学号,它引用了本关系"学号"属性,即"班长"必须是确实存在的学生的学号。

这三个例子说明关系与关系之间存在着相互引用,相互约束的情况。下面先引入外码的概念,然后给出表达关系之间相互引用约束的参照完整性的定义。

定义 2.5 设 F 是基本关系 R 的一个或一组属性,但不是关系 R 的码。K_s 是基本关系 S 的主码。如果 F 与 K_s 相对应,则称 F 是 R 的**外码**(Foreign Key)。并称基本关系 R 为参照关系(Referencing Relation),基本关系 S 为被参照关系(Referenced Relation)或目标关系(Target Relation)。关系 R 和 S 不一定是不同的关系。

$$R(K_r, F, \cdots), S(K_s, \cdots)$$

参照关系 ——→ 被参照关系(目标关系)

显然,目标关系 S 的主码 K_s 和参照关系 R 的外码 F 必须定义在同一个(或同一组)域上。

在[例 1]中,学生关系的"专业号"属性与专业关系的主码"专业号"相对应,因此"专业号"属性是学生关系的外码。这里专业关系是被参照关系,学生关系为参照关系。如图 2.1(a)所示。

在[例 2]中,选修关系的"学号"属性与学生关系的主码"学号"相对应;选修关系的"课程号"属性与课程关系的主码"课程号"相对应,因此"学号"和"课程号"属性是选修关系的外码。这里学生关系和课程关系均为被参照关系,选修关系为参照关系。如图2.1(b)所示。

在[例 3]中,"班长"属性与本身的主码"学号"属性相对应,因此"班长"是外码。这里,

学生关系既是参照关系也是被参照关系。如图2.1(c)所示。

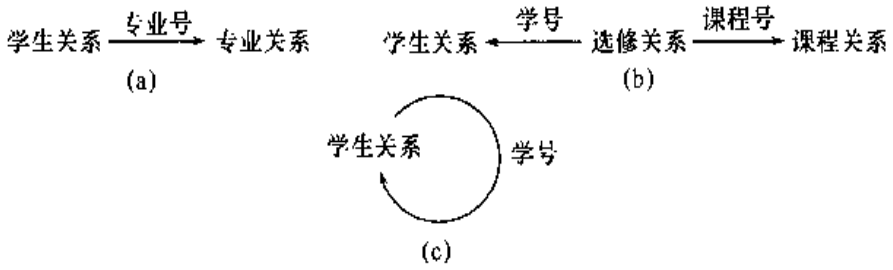


图2.1 关系的参照图

需要指出的是,外码并不一定要与相应的主码同名,如[**例3**]中学生关系的主码名为学号,外码名为班长。不过,在实际应用当中,为了便于识别,当外码与相应的主码属于不同关系时,往往给它们取相同的名字。

参照完整性规则就是定义外码与主码之间的引用规则。

规则2.2 参照完整性规则 若属性(或属性组)F是基本关系R的外码,它与基本关系S的主码K_s相对应(基本关系R和S不一定是不同的关系),则对于R中每个元组在F上的值必须为:

- 或者取空值(F的每个属性值均为空值);
- 或者等于S中某个元组的主码值。

例如,对于[**例1**],学生关系中每个元组的"专业号"属性只能取下面两类值:

(1) 空值,表示尚未给该学生分配专业;

(2) 非空值,这时该值必须是专业关系中某个元组的"专业号"值,表示该学生不可能分配到一个不存在的专业中。即被参照关系"专业"中一定存在一个元组,它的主码值等于该参照关系"学生"中的外码值。

对于[**例2**],按照参照完整性规则,"学号"和"课程号"属性也可以取两类值:空值或目标关系中已经存在的值。但由于"学号"和"课程号"是选修关系中的主属性,按照实体完整性规则,它们均不能取空值。所以选修关系中的"学号"和"课程号"属性实际上只能取相应被参照关系中已经存在的主码值。

参照完整性规则中,R与S可以是同一个关系。例如对于[**例3**],按照参照完整性规则,"班长"属性值可以取两类值:

(1) 空值,表示该学生所在班级尚未选出班长;

(2) 非空值,这时该值必须是本关系中某个元组的学号值。

2.3.4 用户定义的完整性(User-defined Integrity)

任何关系数据库系统都应该支持实体完整性和参照完整性。这是关系模型所要求的。

除此之外,不同的关系数据库系统根据其应用环境的不同,往往还需要一些特殊的约束

条件。用户定义的完整性就是针对某一具体关系数据库的约束条件。它反映某一具体应用所涉及的数据必须满足的语义要求。例如某个属性必须取唯一值、某个非主属性也不能取空值(例如在[**例 1**]的学生关系中必须给出学生姓名,就可以要求学生姓名不能取空值)、某个属性的取值范围在 0 ~ 100 之间(例如学生的成绩)等。

关系模型应提供定义和检验这类完整性的机制,以便用统一的系统的方法处理它们,而不要由应用程序承担这一功能。

在早期的 RDBMS 中没有提供定义和检验这些完整性的机制,因此需要应用开发人员在应用系统的程序中进行检查。例如在[**例 2**]的选修关系中,每插入一条记录,必须在应用程序中写一段程序来检查其中的学号是否等于学生关系中的某个学号,检查其中的课程号是否等于课程关系中的某个课程号。如果等于,则插入这一条选修记录,否则就拒绝插入,并给出错误信息。

2.4 关系代数

关系代数是一种抽象的查询语言,它用对关系的运算来表达查询。

任何一种运算都是将一定的运算符作用于一定的运算对象上,得到预期的运算结果。所以运算对象、运算符、运算结果是运算的三大要素。

关系代数的运算对象是关系,运算结果亦为关系。关系代数用到的运算符包括四类:集合运算符、专门的关系运算符、算术比较符和逻辑运算符,如表 2.4 所示。

表 2.4 关系代数运算符

运算符		含义	运算符		含义
集合运算符	∪	并	比较运算符	>	大于
	−	差		≥	大于等于
	∩	交		<	小于
	×	笛卡尔积		≤	小于等于
				=	等于
				<>	不等于
专门的关系运算符	σ	选择	逻辑运算符	¬	非
	π	投影		∧	与
	⋈	连接		∨	或
	÷	除			

关系代数的运算按运算符的不同可分为传统的集合运算和专门的关系运算两类。

其中传统的集合运算将关系看成元组的集合,其运算是从关系的"水平"方向即行的角度来进行。而专门的关系运算不仅涉及行而且涉及列。比较运算符和逻辑运算符是用来辅

助专门的关系运算符进行操作的。

2.4.1 传统的集合运算

传统的集合运算是二目运算，包括并、差、交、笛卡尔积四种运算。

设关系 R 和关系 S 具有相同的目 n（即两个关系都有 n 个属性），且相应的属性取自同一个域，t 是元组变量，$t \in R$ 表示 t 是 R 的一个元组。

可以定义并、差、交、笛卡尔积运算如下：

1. 并（Union）

关系 R 与关系 S 的并记作

$$R \cup S = \{t \mid t \in R \vee t \in S\}$$

其结果仍为 n 目关系，由属于 R 或属于 S 的元组组成。

2. 差（Except）

关系 R 与关系 S 的差记作

$$R - S = \{t \mid t \in R \wedge t \notin S\}$$

其结果关系仍为 n 目关系，由属于 R 而不属于 S 的所有元组组成。

3. 交（Intersection）

关系 R 与关系 S 的交记作

$$R \cap S = \{t \mid t \in R \wedge t \in S\}$$

其结果关系仍为 n 目关系，由既属于 R 又属于 S 的元组组成。关系的交可以用差来表示，即 $R \cap S = R - (R - S)$。

4. 笛卡尔积（Cartesian Product）

在这里的笛卡尔积严格地讲应该是广义的笛卡尔积（Extended Cartesian Product）。因为这里笛卡尔积的元素是元组。

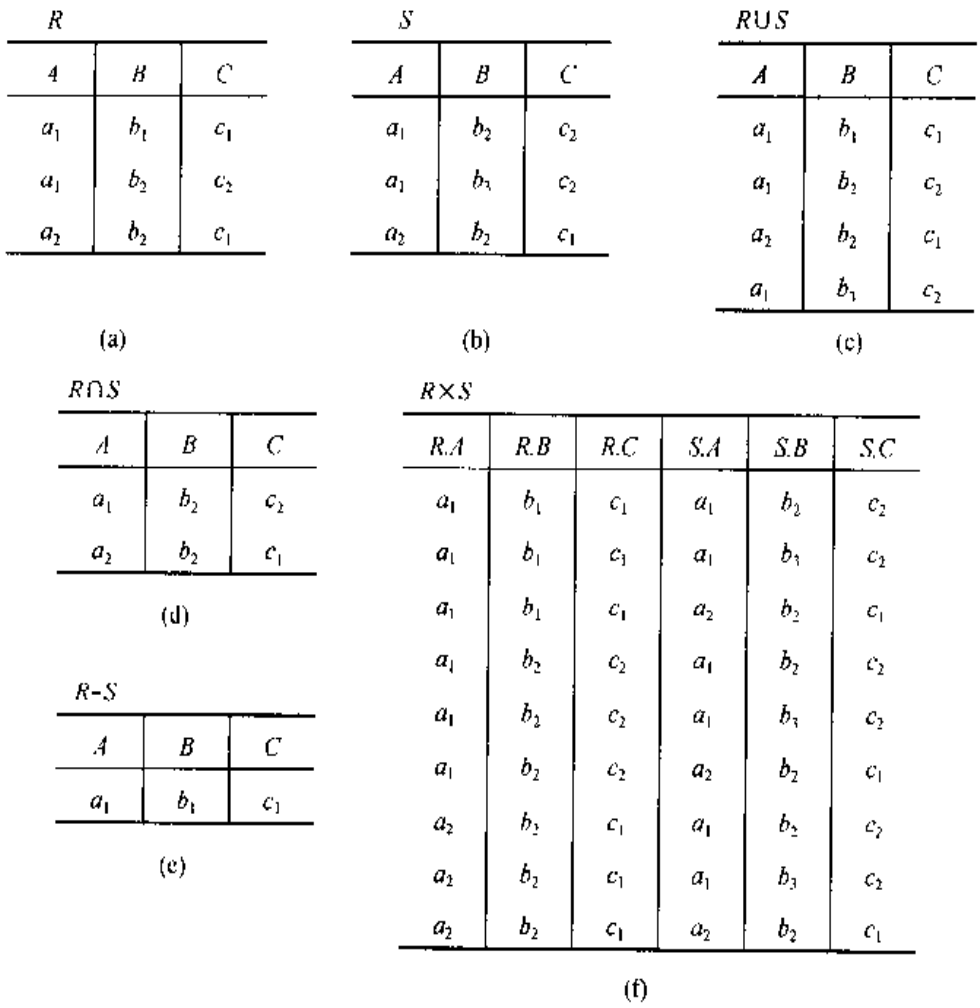
两个分别为 n 目和 m 目的关系 R 和 S 的笛卡尔积是一个 $(n+m)$ 列的元组的集合。元组的前 n 列是关系 R 的一个元组，后 m 列是关系 S 的一个元组。若 R 有 k_1 个元组，S 有 k_2 个元组，则关系 R 和关系 S 的笛卡尔积有 $k_1 \times k_2$ 个元组。记作：

$$R \times S = \{\widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S\}$$

图 2.2(a)、图 2.2(b)分别为具有 3 个属性列的关系 R、S。图 2.2(c)为关系 R 与 S 的并。图 2.2(d)为关系 R 与 S 的交。图 2.2(e)为关系 R 和 S 的差。图 2.2(f)为关系 R 和 S 的笛卡尔积。

2.4.2 专门的关系运算

专门的关系运算包括选择、投影、连接、除运算等。为了叙述上的方便，先引入几个记号。



R

A	*B*	*C*
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_2	c_1

(a)

S

A	*B*	*C*
a_1	b_2	c_2
a_1	b_3	c_2
a_2	b_2	c_1

(b)

R∪*S*

A	*B*	*C*
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_2	c_1
a_1	b_3	c_2

(c)

R∩*S*

A	*B*	*C*
a_1	b_2	c_2
a_2	b_2	c_1

(d)

R-*S*

A	*B*	*C*
a_1	b_1	c_1

(e)

R×*S*

R.A	*R.B*	*R.C*	*S.A*	*S.B*	*S.C*
a_1	b_1	c_1	a_1	b_2	c_2
a_1	b_1	c_1	a_1	b_3	c_2
a_1	b_1	c_1	a_2	b_2	c_1
a_1	b_2	c_2	a_1	b_2	c_2
a_1	b_2	c_2	a_1	b_3	c_2
a_1	b_2	c_2	a_2	b_2	c_1
a_2	b_2	c_1	a_1	b_2	c_2
a_2	b_2	c_1	a_1	b_3	c_2
a_2	b_2	c_1	a_2	b_2	c_1

(f)

图 2.2 传统集合运算举例

(1) 设关系模式为 $R(A_1,A_2,\cdots,A_n)$。它的一个关系设为 R。$t\in R$ 表示 t 是 R 的一个元组。$t[A_i]$ 则表示元组 t 中相应于属性 A_i 的一个分量。

(2) 若 $A=\{A_{i1},A_{i2},\cdots,A_{ik}\}$，其中 $A_{i1},A_{i2},\cdots,A_{ik}$ 是 $A_1,A_2,\cdots,A_n$ 中的一部分，则 A 称为属性列或属性组。$t[A]=(t[A_{i1}],t[A_{i2}],\cdots,t[A_{ik}])$ 表示元组 t 在属性列 A 上诸分量的集合。$\bar{A}$ 则表示 $\{A_1,A_2,\cdots,A_n\}$ 中去掉 $\{A_{i1},A_{i2},\cdots,A_{ik}\}$ 后剩余的属性组。

(3) R 为 n 目关系，S 为 m 目关系。$t_r\in R,t_s\in S$，$\widehat{t_rt_s}$ 称为元组的连接(Concatenation)。它是一个 $n+m$ 列的元组，前 n 个分量为 R 中的一个 n 元组，后 m 个分量为 S 中的一个 m 元组。

(4) 给定一个关系 $R(X,Z)$，X 和 Z 为属性组。当 $t[X]=x$ 时，x 在 R 中的象集(Images Set)定义为：

$$Z_x=\{t[Z]\mid t\in R,t[X]=x\}$$

它表示 R 中属性组 X 上值为 x 的诸元组在 Z 上分量的集合。

例如，图 2.3 中，x_1 在 R 中的象集 $Zx_1=\{Z_1,Z_2,Z_3\}$，

x_2 在 R 中的象集 $Zx_2 = \{Z_2, Z_3\}$，

x_3 在 R 中的象集 $Zx_3 = \{Z_1, Z_3\}$

R

x_1	Z_1
x_1	Z_2
x_1	Z_3
x_2	Z_2
x_2	Z_3
x_3	Z_1
x_3	Z_3

图 2.3 象集举例

下面给出这些专门的关系运算的定义：

1. 选择(Selection)

选择又称为限制(Restriction)。它是在关系 R 中选择满足给定条件的诸元组，记作

$$\sigma_F(R) = \{t \mid t \in R \wedge F(t) = '真'\}$$

其中 F 表示选择条件，它是一个逻辑表达式，取逻辑值“真”或“假”。

逻辑表达式 F 的基本形式为

$$X_1 \theta Y_1$$

其中 θ 表示比较运算符，它可以是 $>$，$\geqslant$，$<$，$\leqslant$，$=$ 或 $<>$。X_1，Y_1 等是属性名，或为常量，或为简单函数；属性名也可以用它的序号来代替。在基本的选择条件上可以进一步进行逻辑运算，即进行求非($\neg$)、与($\wedge$)、或($\vee$)运算。

选择运算实际上是从关系 R 中选取使逻辑表达式 F 为真的元组。这是从行的角度进行的运算。

设有一个学生－课程数据库，包括学生关系 Student、课程关系 Course 和选修关系 SC。如图 2.4 所示。下面的许多例子将对这 3 个关系进行运算。

[例 1] 查询信息系(IS 系)全体学生。

$$\sigma_{Sdept='IS'}(\text{Student}) \text{ 或 } \sigma_{5='IS'}(\text{Student})$$

其中下角标“5”为 Sdept 的属性序号。结果如图 2.5(a)所示。

[例 2] 查询年龄小于 20 岁的学生。

$$\sigma_{Sage<20}(\text{Student}) \text{ 或 } \sigma_{4<20}(\text{Student})$$

结果如图 2.5(b)所示。

2. 投影(Projection)

关系 R 上的投影是从 R 中选择出若干属性列组成新的关系。记作

$$\pi_A(R) = \{t[A] \mid t \in R\}$$

其中 A 为 R 中的属性列。

投影操作是从列的角度进行的运算。

[例 3] 查询学生的姓名和所在系，即求 Student 关系上学生姓名和所在系两个属性上的投影。

$$\pi_{Sname,Sdept}(\text{Student}) \text{ 或 } \pi_{2,5}(\text{Student})$$

结果如图 2.6(a)。

投影之后不仅取消了原关系中的某些列，而且还可能取消某些元组，因为取消了某些属性列后，就可能出现重复行，应取消这些完全相同的行。

Student

学 号 Sno	姓 名 Sname	性 别 Ssex	年 龄 Sage	所在系 Sdept
95001	李勇	男	20	CS
95002	刘晨	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS

(a)

Course

课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL 语言	6	4

(b)

SC

学 号 Sno	课 程 号 Cno	成 绩 Grade
95001	1	92
95001	2	85
95001	3	88
95002	2	90
95002	3	80

(c)

图 2.4 学生 - 课程数据库

Sno	Sname	Ssex	Sage	Sdept
95002	刘晨	女	19	IS
95004	张立	男	19	IS

(a)

Sno	Sname	Ssex	Sage	Sdept
95002	刘晨	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS

(b)

图 2.5 选择运算举例

[例4] 查询学生关系 Student 中都有哪些系，即查询关系 Student 上所在系属性上的投影。

$$\pi_{Sdept}(Student)$$

结果如图 2.6(b)。Student 关系原来有 4 个元组，而投影结果取消了重复的 IS 元组，因此只有 3 个元组。

Sname	Sdept
李勇	CS
刘晨	IS
王敏	MA
张立	IS

(a)

Sdept
CS
IS
MA

(b)

图 2.6 投影运算举例

3. 连接(Join)

连接也称为 θ 连接。它是从两个关系的笛卡尔积中选取属性间满足一定条件的元组。记作

$$R \underset{A\theta B}{\bowtie} S = \{\widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A]\theta t_s[B]\}$$

其中 A 和 B 分别为 R 和 S 上度数相等且可比的属性组。θ 是比较运算符。连接运算从 R 和 S 的笛卡尔积 $R \times S$ 中选取 R 关系在 A 属性组上的值与 S 关系在 B 属性组上值满足比较关系 θ 的元组。

连接运算中有两种最为重要也最为常用的连接，一种是等值连接(equijoin)，另一种是自然连接(Natural join)。

θ 为“=”的连接运算称为**等值连接**。

它是从关系 R 与 S 的广义笛卡尔积中选取 A，B 属性值相等的那些元组，即等值连接为

$$R \underset{A=B}{\bowtie} S = \{\widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A] = t_s[B]\}$$

自然连接(Natural join)是一种特殊的等值连接。它要求两个关系中进行比较的分量必须是相同的属性组，并且在结果中把重复的属性列去掉。即若 R 和 S 具有相同的属性组 B，则自然连接可记作

$$R \bowtie S = \{\widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[B] = t_s[B]\}$$

一般的连接操作是从行的角度进行运算。但自然连接还需要取消重复列，所以是同时从行和列的角度进行运算。

[例5] 设图 2.7(a)和(b)分别为关系 R 和关系 S，图 2.7(c) 为一般连接 $R \underset{C<E}{\bowtie} S$ 的结

果，图 2.7(d) 为等值连接 $R \underset{R.B=S.B}{\bowtie} S$ 的结果，图 2.7(e)为自然连接 $R \bowtie S$ 的结果。

R

A	B	C
a_1	b_1	5
a_1	b_2	6
a_2	b_3	8
a_2	b_4	12

(a) 关系R

S

B	E
b_1	3
b_2	7
b_3	10
b_3	2
b_5	2

(b) 关系S

$R \underset{C\ E}{\bowtie} S$

A	$R.B$	C	$S.B$	E
a_1	b_1	5	b_2	7
a_1	b_1	5	b_3	10
a_1	b_2	6	b_2	7
a_1	b_2	6	b_3	10
a_2	b_3	8	b_3	10

(c) 一般连接

A	$R.B$	C	$S.B$	E
a_1	b_1	5	b_1	3
a_1	b_2	6	b_2	7
a_2	b_3	8	b_3	10
a_2	b_3	8	b_3	2

(d) 等值连接

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2

(e) 自然连接

图 2.7　连接运算举例

两个关系 R 和 S 在做自然连接时，选择两个关系在公共属性上值相等的元组构成新的关系。此时，关系 R 中某些元组有可能在 S 中不存在公共属性上值相等的元组，从而造成 R 中这些元组在操作时被舍弃了，同样，S 中某些元组也可能被舍弃。例如，在图 2.7 的自然连接中，R 中的第 4 个元组，S 中的第 5 个元组都被舍弃掉了。

如果把舍弃的元组也保存在结果关系中，而在其他属性上填空值(Null)，那么这种连接就叫做**外连接**(outer join)。如果只把左边关系 R 中要舍弃的元组保留就叫做**左外连接**(LEFT OUTER JOIN 或 LEFT JOIN)，如果只把右边关系 S 中要舍弃的元组保留就叫做**右外连接**(RIGHT OUTER JOIN 或 RIGHT JOIN)。在图 2.8 中，图(a)是图 2.7 中的关系 R 和关系 S 的外连接，图(b)是左外连接，图(c)是右外连接。

4. 除运算(Division)

给定关系 $R(X,Y)$ 和 $S(Y,Z)$，其中 X、Y、Z 为属性组。R 中的 Y 与 S 中的 Y 可以有不同的属性名，但必须出自相同的域集。

R 与 S 的除运算得到一个新的关系 $P(X)$，P 是 R 中满足下列条件的元组在 X 属性列上的投影：元组在 X 上分量值 x 的象集 Y_x 包含 S 在 Y 上投影的集合。记作

$$R \div S = \{t_r[X] \mid t_r \in R \wedge \pi_Y(S) \subseteq Y_x\}$$

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2
a_2	b_4	12	NULL
NULL	b_5	NULL	2

(a) 外连接

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2
a_2	b_4	12	NULL

(b) 左外连接

A	B	C	E
a_1	b_1	5	3
a_1	b_2	6	7
a_2	b_3	8	10
a_2	b_3	8	2
NULL	b_5	NULL	2

(c) 右外连接

图 2.8 外连接运算举例

其中 Y_x 为 x 在 R 中的象集，$x=t_r[X]$。

除操作是同时从行和列角度进行运算。

[例 6] 设关系 R、S 分别为图 2.9 中的(a)和(b)，$R\div S$ 的结果为图 2.9(c)。

在关系 R 中，A 可以取 4 个值$\{a_1,a_2,a_3,a_4\}$。其中：

a_1 的象集为$\{(b_1,c_2),(b_2,c_3),(b_2,c_1)\}$

a_2 的象集为$\{(b_3,c_7),(b_2,c_3)\}$

a_3 的象集为$\{(b_4,c_6)\}$

a_4 的象集为$\{(b_6,c_6)\}$

S 在(B,C)上的投影为$\{(b_1,c_2),(b_2,c_1),(b_2,c_3)\}$

显然只有 a_1 的象集$(B,C)_{a_1}$包含了 S 在(B,C)属性组上的投影，所以

$R\div S=\{a_1\}$

R

A	B	C
a_1	b_1	c_2
a_2	b_3	c_7
a_3	b_4	c_6
a_1	b_2	c_3
a_4	b_6	c_6
a_2	b_2	c_3
a_1	b_2	c_1

(a)

S

B	C	D
b_1	c_2	d_1
b_2	c_1	d_1
b_2	c_3	d_2

(b)

$R\div S$

A
a_1

(c)

图 2.9 除运算举例

下面再以学生－课程数据库为例，给出几个综合应用多种关系代数运算进行查询的例子。

[例 7] 查询至少选修 1 号课程和 3 号课程的学生号码。

首先建立一个临时关系 K：

K
Cno
1
3

然后求：$\pi_{Sno,Cno}(SC) \div K$

结果为 {95001}。

求解过程与[例 6]类似，先对 SC 关系在(Sno,Cno)属性上投影，然后逐一求出每一学生(Sno)的象集，并依次检查这些象集是否包含 K。

[例 8] 查询选修了 2 号课程的学生的学号。

$$\pi_{Sno}(\sigma_{Cno='2'}(SC)) = \{95001,95002\}$$

[例 9] 查询至少选修了一门其直接先行课为 5 号课程的学生姓名。

$$\pi_{Sname}(\sigma_{Cpno='5'}(Course) \bowtie SC \bowtie \pi_{Sno,Sname}(Student))$$

或

$$\pi_{Sname}(\pi_{Sno}(\sigma_{Cpno='5'}(Course) \bowtie SC) \bowtie \pi_{Sno,Sname}(Student))$$

[例 10] 查询选修了全部课程的学生号码和姓名。

$$\pi_{Sno,Cno}(SC) \div \pi_{Cno}(Course) \bowtie \pi_{Sno,Sname}(Student)$$

本节介绍了 8 种关系代数运算，其中并、差、笛卡尔积、选择和投影 5 种运算为基本的运算。其他 3 种运算，即交、连接和除，均可以用这 5 种基本运算来表达。引进它们并不增加语言的能力，但可以简化表达。

关系代数中，这些运算经有限次复合后形成的式子称为关系代数表达式。

2.5 关系演算

关系演算是以数理逻辑中的谓词演算为基础的。按谓词变元的不同，关系演算可分为元组关系演算和域关系演算。本节先介绍元组关系演算，然后简单介绍域关系演算。

2.5.1 元组关系演算语言 ALPHA

元组关系演算以元组变量作为谓词变元的基本对象。一种典型的元组关系演算语言是 E. F. Codd 提出的 ALPHA 语言。这一语言虽然没有实际实现，但关系数据库管理系统 INGRES 最初所用的 QUEL 语言是参照 ALPHA 语言研制的，与 ALPHA 十分类似。

ALPHA 语言主要有 GET, PUT, HOLD, UPDATE, DELETE, DROP 6 条语句，语句的基本

格式是

操作语句 工作空间名(表达式):操作条件

其中表达式用于指定语句的操作对象,它可以是关系名或(和)属性名,一条语句可以同时操作多个关系或多个属性。操作条件是一个逻辑表达式,用于将操作结果限定在满足条件的元组中,操作条件可以为空。除此之外,还可以在基本格式的基础上加上排序要求、定额要求等。

一、检索操作

检索操作用 GET 语句实现。

(1) 简单检索(即不带条件的检索)

[例 1] 查询所有被选修的课程号码。

GET W (SC. Cno)

W 为工作空间名。这里条件为空,表示没有限定条件。

[例 2] 查询所有学生的数据。

GET W (Student)

(2) 限定的检索(即带条件的检索)

[例 3] 查询信息系(IS)中年龄小于 20 岁的学生的学号和年龄。

GET W (Student. Sno, Student. Sage): Student. Sdept = 'IS' ∧ Student. Sage < 20

(3) 带排序的检索

[例 4] 查询计算机科学系(CS)学生的学号、年龄,结果按年龄降序排序。

GET W (Student. Sno, Student. Sage): Student. Sdept = 'CS' DOWN Student. Sage

DOWN 表示降序排序。

(4) 带定额的检索

[例 5] 取出一个信息系学生的学号。

GET W (1) (Student. Sno): Student. Sdept = 'IS'

排序和定额可以一起使用。所谓带定额的检索是指规定了检索出元组的个数,方法是在 W 后括号中加上定额数量。

[例 6] 查询信息系年龄最大的三个学生的学号及其年龄,结果按年龄降序排序。

GET W (3) (Student. Sno, Student. Sage): Student. Sdept = 'IS' DOWN Student. Sage

(5) 用元组变量的检索

前面已讲到,元组关系演算是以元组变量作为谓词变元的基本对象。元组变量是在某一关系范围内变化(所以也称为范围变量 Range Variable),一个关系可以设多个元组变量。

元组变量主要有两方面的用途:

① 简化关系名。如果关系的名字很长,使用起来就会感到不方便,这时可以设一个较短名字的元组变量来代替关系名。

② 操作条件中使用量词时必须用元组变量。

[例 7] 查询信息系学生的名字。

```
RANGE Student X
GET W (X.Sname): X.Sdept = 'IS'
```

ALPHA 语言用 RANGE 来说明元组变量。本例中 X 是关系 Student 上的元组变量,用途是简化关系名,即用 X 代表 Student。

(6) 用存在量词的检索

操作条件中使用量词时必须用元组变量。

[例 8] 查询选修 2 号课程的学生名字。

```
RANGE SC X
GET W (Student.Sname): ∃X(X.Sno = Student.Sno ∧ X.Cno = '2')
```

[例 9] 查询选修了这样课程的学生学号,其直接先行课是 6 号课程。

```
RANGE Course CX
GET W (SC.Sno): ∃CX (CX.Cno = SC.Cno ∧ CX.Pcno = '6')
```

[例 10] 查询至少选修一门其先行课为 6 号课程的学生名字。

```
RANGE Course CX
      SC     SCX
GET W (Student.Sname): ∃SCX(SCX.Sno = Student.Sno ∧
                        ∃CX (CX.Cno = SCX.Cno ∧ CX.Pcno = '6'))
```

在[例 10]中的元组关系演算公式可以变换为前束范式(Prenex normal form)的形式:

```
GET W (Student.Sname): ∃SCX ∃CX (SCX.Sno = Student.Sno ∧
                                 CX.Cno = SCX.Cno ∧ CX.Pcno = '6')
```

[例 8]、[例 9]、[例 10]中的元组变量都是为存在量词而设的。其中[例 10]需要对两个关系作用存在量词,所以设了两个元组变量。

(7) 带有多个关系的表达式的检索

上面所举的各个例子中,虽然查询时可能会涉及多个关系,即公式中可能涉及多个关系,但查询结果表达式中只有一个关系。实际上表达式中是可以有多个关系的。

[例 11] 查询成绩为 90 分以上的学生名字与课程名字。

本查询所要求的结果是学生名字和课程名字,分别在 Student 和 Course 两个关系中。

```
RANGE SC SCX
GET W (Student.Sname, Course.Cname): ∃SCX (SCX.Grade ≥ 90 ∧
                        SCX.Sno = Student.Sno ∧ Course.Cno = SCX.Cno)
```

(8) 用全称量词的检索

[例 12] 查询不选 1 号课程的学生名字。

```
RANGE SC SCX
GET W (Student.Sname):∀SCX (SCX.Sno≠Student.Sno∨SCX.Cno≠'1')
```

本例也可以用存在量词来表示：

```
RANGE SC SCX
GET W (Student.Sname):¬∃SCX (SCX.Sno=Student.Sno∧SCX.Cno='1')
```

(9) 用两种量词的检索

[例 13] 查询选修了全部课程的学生姓名。

```
RANGE Course CX
      SC     SCX
GET W (Student.Sname):∀CX∃SCX (SCX.Sno=Student.Sno∧
                               SCX.Cno=CX.Cno)
```

(10) 用蕴涵(Implication)的检索

[例 14] 查询最少选修了 95002 学生所选课程的学生学号。

本例题的求解思路是，对 Course 中的所有课程，依次检查每一门课程，看 95002 是否选修了该课程，如果选修了，则再看某一个学生是否也选修了该门课。如果对于 95002 所选的每门课程该学生都选修了，则该学生为满足要求的学生。把所有这样的学生全都找出来即完成了本题。

```
RANGE Course CX
      SC     SCX (注意,这里 SC 设了两个元组变量)
      SC     SCY
GET W (Student.Sno):∀CX(∃SCX (SCX.Sno='95002'∧SCX.Cno=CX.Cno)
                        ⇒∃SCY(SCY.Sno=Student.Sno∧SCY.Cno=CX.Cno))
```

(11) 聚集函数

用户在使用查询语言时，经常要作一些简单的计算，例如要求符合某一查询要求的元组数，求某个关系中所有元组在某属性上的值的总和或平均值等。为了方便用户，关系数据语言中建立了有关这类运算的标准函数库供用户选用。这类函数通常称为聚集函数或内部函数(Built-in function)。关系演算中提供了 COUNT, TOTAL, MAX, MIN, AVG 等聚集函数，其含义如表 2.5 所示。

表 2.5 关系演算中的聚集函数

函数名	功能
COUNT	对元组计数
TOTAL	求总和
MAX	求最大值
MIN	求最小值
AVG	求平均值

[例 15] 查询学生所在系的数目。

```
GET W (COUNT(Student.Sdept))
```

COUNT 函数在计数时会自动排除重复值。

[例 16] 查询信息系学生的平均年龄。

```
GET W (AVG(Student. Sage); Student. Sdept = 'IS')
```

二、更新操作

(1) 修改操作

修改操作用 UPDATE 语句实现。其步骤是:

① 首先用 HOLD 语句将要修改的元组从数据库中读到工作空间中;

② 然后用宿主语言修改工作空间中元组的属性值;

③ 最后用 UPDATE 语句将修改后的元组送回数据库中。

需要注意的是,单纯检索数据使用 GET 语句即可,但为修改数据而读元组时必须使用 HOLD 语句,HOLD 语句是带上并发控制的 GET 语句。有关并发控制的概念将在第 11 章中详细介绍。

[例 17] 把 95007 学生从计算机科学系转到信息系。

```
HOLD W (Student. Sno, Student Sdept); Student. Sno = '95007'
                                    (从 Student 关系中读出 95007 学生的数据)
MOVE 'IS' TO W. Sdept               (用宿主语言进行修改)
UPDATE W                            (把修改后的元组送回 Student 关系)
```

在该例中用 HOLD 语句来读 95007 的数据,而不是用 GET 语句。

如果修改操作涉及到两个关系的话,就要执行两次 HOLD-MOVE-UPDATE 操作序列。

在 ALPHA 语言中,修改关系主码的操作是不允许的,例如不能用 UPDATE 语句将学号 95001 改为 95102。如果需要修改主码值,只能先用删除操作删除该元组,然后再把具有新主码值的元组插入到关系中。

(2) 插入操作

插入操作用 PUT 语句实现。其步骤是:

① 首先用宿主语言在工作空间中建立新元组;

② 然后用 PUT 语句把该元组存入指定的关系中。

[例 18] 学校新开设了一门 2 学分的课程"计算机组织与结构",其课程号为 8,直接先行课为 6 号课程。插入该课程元组。

```
MOVE '8' TO W. Cno
MOVE '计算机组织与结构' TO W. Cname
MOVE '6' TO W. Cpno
MOVE '2' TO W. Ccredit
PUT W (Course)                      (把 W 中的元组插入指定关系 Course 中)
```

PUT 语句只对一个关系操作,也就是说表达式必须为单个关系名。

(3) 删除

删除操作用 DELETE 语句实现。其步骤为:

① 用 HOLD 语句把要删除的元组从数据库中读到工作空间中；

② 用 DELETE 语句删除该元组。

[例 19] 95110 学生因故退学，删除该学生元组。

```
HOLD W (Student): Student.Sno ='95110'
DELETE W
```

[例 20] 将学号 95001 改为 95102。

```
HOLD W (Student): Student.Sno ='95001'
DELETE W
MOVE '95102' TO W.Sno
MOVE '李勇' TO W.Sname
MOVE '男' TO W.Ssex
MOVE '20' TO W.Sage
MOVE 'CS' TO W.Sdept
PUT W (Student)
```

[例 21] 删除全部学生。

```
HOLD W (Student)
DELETE W
```

由于 SC 关系与 Student 关系之间的具有参照关系，为保证参照完整性，删除 Student 中元组时相应地要删除 SC 中的元组(手工删除或由 DBMS 自动执行)：

```
HOLD W (SC)
DELETE W
```

*2.5.2 元组关系演算

上一节讲解了一种具体的元组关系演算语言，这一节讲解抽象的元组关系演算。

为了讨论方便，先允许关系(的基数)是无限的。然后再对这种情况下定义的演算作适当的修改，保证关系演算中的每一个公式表示的是有限关系。

在元组关系演算系统中，称

$$\{t \mid \phi(t)\}$$

为**元组演算表达式**。其中 t 是元组变量，$\phi(t)$ 为**元组关系演算公式**简称**公式**。它由原子公式和运算符组成。

原子公式有三类：

1. $R(t)$

R 是关系名，t 是元组变量。$R(t)$ 表示 t 是 R 中的元组。于是，关系 R 可表示为

$$\{t \mid R(t)\}$$

2. $t[i]\theta u[j]$

t 和 u 是元组变量，θ 是算术比较运算符。$t[i]\theta u[j]$ 表示断言“元组 t 的第 i 个分量与元

组 u 的第 j 个分量满足比较关系 θ"。例如,$t[2] < u[3]$ 表示元组 t 的第2个分量小于元组 u 的第3个分量。

3. $t[i]\theta c$ 或 $c\theta t[i]$

这里 c 是常量,该公式表示"t 的第 i 个分量与常量 c 满足比较关系 θ"。例如,$t[4] = 3$ 表示元组 t 的第4个分量等于3。

在关系演算中定义了"自由元组变量"和"约束元组变量"的概念。这些概念和谓词演算中的概念完全一样。若公式中的一个元组变量前有"全称量词"或"存在量词",则称该变量为约束元组变量,否则称自由元组变量。

公式可以递归定义如下:

(1) 每个原子公式是公式。

(2) 如果 ϕ_1 和 ϕ_2 是公式,则 $\phi_1 \wedge \phi_2$,$\phi_1 \vee \phi_2$,$\neg \phi_1$ 也是公式。分别表示:

- 如果 ϕ_1 和 ϕ_2 同时为真,则 $\phi_1 \wedge \phi_2$ 才为真,否则为假;
- 如果 ϕ_1 和 ϕ_2 中一个或同时为真,则 $\phi_1 \vee \phi_2$ 为真,仅当 ϕ_1 和 ϕ_2 同时为假时,$\phi_1 \vee \phi_2$ 才为假;
- 若 ϕ_1 为真,则 $\neg \phi_1$ 为假。

(3) 若 ϕ 是公式,则 $\exists t(\phi)$ 也是公式。其中符号 ∃ 是存在量词符号,$\exists t(\phi)$ 表示:若有一个 t 使 ϕ 为真,则 $\exists t(\phi)$ 为真,否则 $\exists t(\phi)$ 为假。

(4) 若 ϕ 是公式,则 $\forall t(\phi)$ 也是公式。其中符号 ∀ 是全称量词符号,$\forall t(\phi)$ 表示:如果对所有 t,都使 ϕ 为真,则 $\forall t(\phi)$ 为真,否则 $\forall t(\phi)$ 为假。

(5) 在元组演算公式中,各种运算符的优先次序为:

① 算术比较运算符最高;

② 量词次之,且 ∃ 的优先级高于 ∀ 的优先级;

③ 逻辑运算符最低,且 ¬ 的优先级高于 ∧ 的优先级,∧ 的优先级高于 ∨ 的优先级;

④ 加括号时,括号中运算符优先,同一括号内的运算符之优先级遵循 ①,②,③ 各项。

(6) 有限次地使用上述五条规则得到的公式是元组关系演算公式,其他公式不是元组关系演算公式。

一个元组演算表达式 $\{t \mid \phi(t)\}$ 表示了使 $\phi(t)$ 为真的元组集合。

关系代数的运算均可以用关系演算表达式来表示(反之亦然)。下面用关系演算表达式来表示五种基本运算:

(1) 并

$$R \cup S = \{t \mid R(t) \vee S(t)\}$$

(2) 差

$$R - S = \{t \mid R(t) \wedge \neg S(t)\}$$

(3) 笛卡尔积

$$R \times S = \{t^{(n+m)} \mid (\exists u^{(n)})(\exists v^{(m)})(R(u) \wedge S(v) \wedge t[1] = u[1] \wedge \cdots \wedge t[n] = u[n] \wedge t[n+1] = v[1] \wedge \cdots \wedge t[n+m] = v[m])\}$$

这里 $t^{(n+m)}$ 表示 t 的目数是$(n+m)$。

(4) 投影

$$\pi_{i1,i2,\cdots,ik}(R) = \{t^{(k)} \mid (\exists u)(R(u) \wedge t[1] = u[i_1] \wedge \cdots \wedge t[k] = u[i_k])\}$$

(5) 选择

$$\sigma_F(R) = \{t \mid R(t) \wedge F'\}$$

F'是公式 F 用 $t[i]$ 代替运算对象 i 得到的等价公式。

下面用关系演算来对图 2.4 学生 - 课程数据库进行查询。

[例 1] 查询信息系(IS 系)全体学生。

$$S_{IS} = \{t \mid \text{Student}(t) \wedge t[5] = \text{'IS'}\}$$

[例 2] 查询年龄小于 20 岁的学生。

$$S_{20} = \{t \mid \text{Student}(t) \wedge t[4] < 20\}$$

[例 3] 查询学生的姓名和所在系。

$$S_1 = \{t^{(2)} \mid (\exists u)(\text{Student}(u) \wedge t[1] = u[2] \wedge t[2] = u[5])\}$$

上面定义的关系演算允许出现无限关系。例如,$\{t \mid \neg R(t)\}$表示所有不属于 R 的元组(元组的目数等于 R 的目数)。要求出这些可能的元组是做不到的,所以必须排除这类无意义的表达式。把不产生无限关系的表达式称为安全表达式,所采取的措施称为安全限制。安全限制通常是定义一个有限的符号集 $\text{dom}(\phi)$,$\text{dom}(\phi)$一定包括出现在 ϕ 以及中间结果和最后结果的关系中的所有符号(实际上是各列中值的汇集)。$\text{dom}(\phi)$不必是最小集。

当满足下列条件时,元组演算表达式$\{t \mid \phi(t)\}$是安全的:

(1) 如果 t 使 $\phi(t)$为真,则 t 的每个分量是 $\text{dom}(\phi)$中的元素。

(2) 对于 ϕ 中每一个形如$(\exists u)(W(u))$的子表达式,若 u 使 $W(u)$为真,则 u 的每个分量是 $\text{dom}(\phi)$中的元素。

(3) 对于 ϕ 中每一个形如$(\forall u)(W(u))$的子表达式,若 u 使 $W(u)$为假,则 u 的每个分量必属于 $\text{dom}(\phi)$。换言之,若 u 某一分量不属于 $\text{dom}(\phi)$,则 $W(u)$为真。

[例 4] 设有关系 R 如图 2.10(a),$S = \{t \mid \neg R(t)\}$,若不进行安全限制,则可能是一个无限关系。所以定义

$$\text{dom}(\phi) = \pi_A(R) \cup \pi_B(R) \cup \pi_C(R) = \{\{a_1, a_2\}, \{b_1, b_2\}, \{c_1, c_2\}\}$$

则 S 是 $\text{dom}(\phi)$中各域值中元素的笛卡尔积与 R 的差集。结果如图 2.10(b)。注意,在做笛卡尔积时各个域中的元素不能搞混。

R

A	B	C
a_1	b_1	c_1
a_2	b_7	c_2

(a)

S

A	B	C
a_1	b_1	c_2
a_1	b_2	c_1
a_1	b_2	c_2
a_2	b_1	c_1
a_2	b_1	c_2
a_2	b_2	c_1

(b)

图 2.10 关系演算安全限制示例

2.5.3 域关系演算语言 QBE

关系演算的另一种形式是域关系演算。域关系演算以元组变量的分量即域变量作为谓词变元的基本对象。1975 年由 M. M. Zloof 提出的 QBE 就是一个很有特色的域关系演算语言,该语言于 1978 年在 IBM370 上得以实现。QBE 也指此关系数据库管理系统。

QBE 是 Query By Example(即通过例子进行查询)的简称,其最突出的特点是它的操作方式。它是一种高度非过程化的基于屏幕表格的查询语言,用户通过终端屏幕编辑程序以填写表格的方式构造查询要求,而查询结果也是以表格形式显示,因此非常直观,易学易用。

QBE 中用示例元素来表示查询结果可能的情况,示例元素实质上就是域变量。QBE 操作框架如图 2.11 所示。

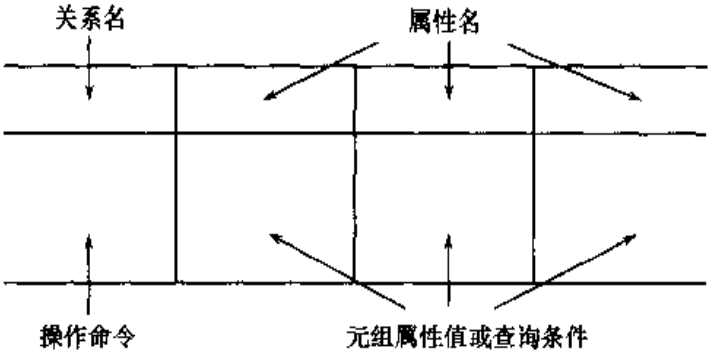


图 2.11 QBE 操作框架

下面以学生 - 课程关系数据库为例,说明 QBE 的用法。

一、检索操作

1. 简单查询

[例 1] 求信息系全体学生的姓名。

操作步骤为:

(1) 用户提出要求;

(2) 屏幕显示空白表格;

(3) 用户在最左边一栏输入关系名 Student;

Student					

(4) 系统显示该关系的属性名;

Student	Sno	Sname	Ssex	Sage	Sdept

(5) 用户在上面构造查询要求。

Student	Sno	Sname	Ssex	Sage	Sdept
		P. T			IS

这里 T 是示例元素,即域变量。QBE 要求示例元素下面一定要加下划线。IS 是查询条件,不用加下划线。P. 是操作符,表示打印(Print),实际上是显示。

查询条件中可以使用比较运算符 >, ≥, <, ≤, = 和 ≠。其中 = 可以省略。

示例元素是这个域中可能的一个值,它不必是查询结果中的元素。比如要求信息系的学生,只要给出任意的一个学生名即可,而不必真是信息系的某个学生名。

对于[**例 1**],可如下构造查询要求:

Student	Sno	Sname	Ssex	Sage	Sdept
		P. 李勇			IS

这里的查询条件是 Sdept = 'IS',其中" = "被省略。

(6) 屏幕显示查询结果

Student	Sno	Sname	Ssex	Sage	Sdept
		李勇 张立			IS

根据用户的查询要求,求出了信息系的学生姓名。

[**例 2**] 查询全体学生的全部数据。

Student	Sno	Sname	Ssex	Sage	Sdept
	P. 95001	P. 李勇	P. 男	P. 20	P. CS

显示全部数据也可以简单地把 P. 操作符作用在关系名上。因此本查询也可以简单地表示如下：

Student	Sno	Sname	Ssex	Sage	Sdept
P.					

2. 条件查询

[例 3] 求年龄大于 19 岁的学生的学号。

Student	Sno	Sname	Ssex	Sage	Sdept
	P. 95001			> 19	

[例 4] 求计算机科学系年龄大于 19 岁的学生的学号。

本查询的条件是 Sdept = 'CS'和 Sage > 19 两个条件的“与”。在 QBE 中，表示两个条件的“与”有两种方法：

(1) 把两个条件写在同一行上；

Student	Sno	Sname	Ssex	Sage	Sdept
	P. 95001			> 19	CS

(2) 把两个条件写在不同行上，但使用相同的示例元素值。

Student	Sno	Sname	Ssex	Sage	Sdept
	P. 95001				CS
	P. 95001			> 19	

[例 5] 查询计算机科学系或者年龄大于 19 岁的学生的学号。

本查询的条件是 Sdept = 'CS'和 Sage > 19 两个条件的“或”。在 QBE 中把两个条件写在不同行上，并且使用不同的示例元素值，即表示条件的“或”。

Student	Sno	Sname	Ssex	Sage	Sdept
	P. 95001				CS
	P. 95002			> 19	

对于多行条件的查询，如[例 4](2)和[例 5]，先输入哪一行是任意的，不影响查询结果。这就允许查询者以不同的思考方式进行查询，十分灵活、自由。

[例 6] 查询既选修了 1 号课程又选修了 2 号课程的学生的学号。

本查询条件是在一个属性中的“与”关系，它只能用“与”条件的第(2)种方法表示，即写两行，但示例元素相同。

SC	Sno	Cno	Grade
	P.95001	1	
	P.95001	2	

[例 7] 查询选修 1 号课程的学生姓名。

本查询涉及两个关系:SC 和 Student。在 QBE 中实现这种查询的方法是通过相同的连接属性值把多个关系连接起来。

Student	Sno	Sname	Ssex	Sage	Sdept
	95001	P.李勇			

SC	Sno	Cno	Grade
	95001	1	

这里示例元素 Sno 是连接属性,其值在两个表中要相同。

[例 8] 查询未选修 1 号课程的学生姓名

这里的查询条件中用到逻辑非。在 QBE 中表示逻辑非的方法是将逻辑非写在关系名下面。

Student	Sno	Sname	Ssex	Sage	Sdept
	95001	P.李勇			

SC	Sno	Cno	Grade
¬	95001	1	

这个查询就是显示学号为95001 的学生名字,而该学生选修 1 号课程的情况为假。

[例 9] 查询有两个人以上选修的课程号。

本查询是在一个表内连接。

这个查询就是要显示这样的课程1,它不仅被95001 选修,而且也被另一个学生(¬ 95001)选修了。

SC	Sno	Cno	Grade
	95001	P.1	
	¬ 95001	1	

3. 聚集函数

为了方便用户,QBE 提供了一些聚集函数,主要包括 CNT,SUM,AVG,MAX,MIN 等,其

含义如表2.6所示。

表2.6 QBE中的聚集函数

函数名	功能
CNT	对元组计数
SUM	求总和
AVG	求平均值
MAX	求最大值
MIN	求最小值

[例10] 查询信息系学生的平均年龄。

Student	Sno	Sname	Ssex	Sage	Sdept
				P.AVG.ALL.	IS

4. 对查询结果排序

对查询结果按某个属性值的升序排序,只需在相应列中填入"AO.",按降序排序则填"DO."。如果按多列排序,用"AO(i)."或"DO(i)."表示,其中i为排序的优先级,i值越小,优先级越高。

[例11] 查全体男生的姓名,要求查询结果按所在系升序排序,对相同系的学生按年龄降序排序。

Student	Sno	Sname	Ssex	Sage	Sdept
		P.李勇	男	DO(2).	AO(1).

二、更新操作

1. 修改操作

修改操作符为"U."。在QBE中,关系的主码不允许修改,如果需要修改某个元组的主码,只能先删除该元组,然后再插入新的主码的元组。

[例12] 把95001学生的年龄改为18岁。

这是一个简单修改操作,不包含算术表达式,因此可以有两种表示方法。

(1) 将操作符"U."放在值上;

Student	Sno	Sname	Ssex	Sage	Sdept
	95001			U.18	

(2) 将操作符"U."放在关系上。

Student	Sno	Sname	Ssex	Sage	Sdept
U.	95001			18	

这里，码 95001 标明要修改的元组。“U.”标明所在的行是修改后的新值。由于主码是不能修改的，所以即使在第二种写法中，系统也不会混淆要修改的属性。

[例 13] 把 95001 学生的年龄增加 1 岁。

这个修改操作涉及表达式，所以只能将操作符“U.”放在关系上。

Student	Sno	Sname	Ssex	Sage	Sdept
	95001			17	
U.	95001			17 + 1	

[例 14] 将计算机科学系所有学生的年龄都增加 1 岁。

Student	Sno	Sname	Ssex	Sage	Sdept
	95008			18	CS
U.	95008			18 + 1	

2. 插入操作

插入操作符为“I.”。新插入的元组必须具有码值，其他属性值可以为空。

[例 15] 把信息系女生 95701，姓名张三，年龄 17 岁存入数据库中。

Student	Sno	Sname	Ssex	Sage	Sdept
I.	95701	张三	女	17	IS

3. 删除操作

删除操作符为“D.”。

[例 16] 删除学生 95089。

Student	Sno	Sname	Ssex	Sage	Sdept
D.	95089				

由于 SC 关系与 Student 关系之间具有参照关系，为保证参照完整性，删除 95089 学生后，通常还应删除 95089 学生选修的全部课程。

SC	Sno	Cno	Grade
D.	95089		

2.6 小 结

关系数据库系统是本书的重点。这是因为关系数据库系统是目前使用最广泛的数据库系统。20世纪70年代以后开发的数据库管理系统产品几乎都是基于关系的。在数据库发展的历史上,最重要的成就之一是关系模型。

关系数据库系统与非关系数据库系统的区别是,关系系统只有"表"这一种数据结构;而非关系数据库系统还有其他数据结构,以及对这些数据结构的操作。

本章系统讲解了关系数据库的重要概念,包括关系模型的数据结构、关系的三类完整性以及关系操作。介绍了用代数方式和逻辑方式来表达的关系语言即关系代数、元组关系演算和域关系演算。这一章从具体到抽象,先讲解了实际的语言ALPHA(元组关系演算语言)和QBE(域关系演算语言),然后讲解了抽象的元组关系演算。

习 题

1. 试述关系模型的3个组成部分。
2. 试述关系数据语言的特点和分类。
3. 定义并理解下列术语,说明它们之间的联系与区别:
 (1) 域,笛卡尔积,关系,元组,属性
 (2) 主码,候选码,外部码
 (3) 关系模式,关系,关系数据库
4. 试述关系模型的完整性规则。在参照完整性中,为什么外部码属性的值也可以为空?什么情况下才可以为空?
5. 设有一个SPJ数据库,包括S、P、J、SPJ 4个关系模式:

 S(SNO,SNAME,STATUS,CITY);
 P(PNO,PNAME,COLOR,WEIGHT);
 J(JNO,JNAME,CITY);
 SPJ(SNO,PNO,JNO,QTY);

供应商表S由供应商代码(SNO)、供应商姓名(SNAME)、供应商状态(STATUS)、供应商所在城市(CITY)组成;

零件表P由零件代码(PNO)、零件名(PNAME)、颜色(COLOR)、重量(WEIGHT)组成;

工程项目表J由工程项目代码(JNO)、工程项目名(JNAME)、工程项目所在城市(CITY)组成;

供应情况表SPJ由供应商代码(SNO)、零件代码(PNO)、工程项目代码(JNO)、供应数量(QTY)组成,表示某供应商 供应某种零件 给某工程项目的数量为QTY。

今有若干数据如下:

S 表

SNO	SNAME	STATUS	CITY
S1	精 益	20	天津
S2	盛 锡	10	北京
S3	东方红	30	北京
S4	丰泰盛	20	天津
S5	为 民	30	上海

P 表

PNO	PNAME	COLOR	WEIGHT
P1	螺 母	红	12
P2	螺 栓	绿	17
P3	螺丝刀	蓝	14
P4	螺丝刀	红	14
P5	凸 轮	蓝	40
P6	齿 轮	红	30

J 表

JNO	JNAME	CITY
J1	三 建	北京
J2	一 汽	长春
J3	弹 簧 厂	天津
J4	造 船 厂	天津
J5	机 车 厂	唐山
J6	无线电厂	常州
J7	半导体厂	南京

SPJ 表

SNO	PNO	JNO	QTY
S1	P1	J1	200
S1	P1	J3	100
S1	P1	J4	700
S1	P2	J2	100
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J4	500
S2	P3	J5	400
S2	P5	J1	400
S2	P5	J2	100
S3	P1	J1	200
S3	P3	J1	200
S4	P5	J1	100
S4	P6	J3	300
S4	P6	J4	200
S5	P2	J4	100
S5	P3	J1	200
S5	P6	J2	200
S5	P6	J4	500

试用关系代数、ALPHA 语言、QBE 语言完成如下查询：

(1) 求供应工程 J1 零件的供应商号码 SNO；

(2) 求供应工程 J1 零件 P1 的供应商号码 SNO；

(3) 求供应工程 J1 零件为红色的供应商号码 SNO；

(4) 求没有使用天津供应商生产的红色零件的工程号 JNO；

(5) 求至少用了供应商 S1 所供应的全部零件的工程号 JNO。

6. 试述等值连接与自然连接的区别和联系。

7. 关系代数的基本运算有哪些？如何用这些基本运算来表示其他运算？

实验1 认识DBMS

一、实验目的

1. 通过对某个商用数据库管理系统的安装使用，初步了解DBMS的工作环境和系统构架。在此推荐金仓数据库KingbaseES，该系统可以从人大金仓公司的网站http://www.kingbase.com.cn/上免费下载。

2. 熟悉对DBMS的安装。

3. 搭建今后实验的平台。

二、实验平台

操作系统：Windows 2000或者Windows XP。

数据库管理系统：KingbaseES或其他国产数据库管理系统，国外的MS SQL Server，Oracle，DB2等。选择安装各个数据库管理系统之前，请仔细看清硬件的配置要求。

三、实验内容及要求

1. 根据安装文件的说明安装数据库管理系统。在安装过程中记录安装的选择，并且对所作的选择进行思考，为何要进行这样的配置，对今后运行数据库管理系统会有什么影响。

2. 学会启动和停止数据库服务。

3. 初步了解RDBMS的安全性，这里主要是用户的登录和服务器预定义角色。可以尝试建立一个新的用户，赋予其数据库管理员的角色，今后的实验可以用该用户来创建数据库应用。

4. 了解RDBMS的系统构架。

5. 了解RDBMS的管理和使用。例如，了解KingbaseES如何通过它提供的工具对数据和数据库服务器进行管理和使用的。学会运用控制管理器和企业管理器进行操作。初步了解KingbaseES的主要工具和它们的功能，为今后的实验做准备。

四、实验报告

《数据库系统概论》实验报告				
题目：	姓名		日期	
实验环境：				
实验内容与完成情况：				
出现的问题：				
解决方案（列出遇到的问题和解决办法，列出没有解决的问题）：				

说明：在教学网站http://www.chinadb.org或http://4A.hep.com.cn上给出了本实验的详细要求和实验报告示例，即Kingbase ES安装和启动。供读者学习参考。

本章参考文献

1 Codd E F. A Relational Model of Data for Large Shared Data Banks. CACM 13 No.6，1970

（1970年，E. F. Codd在[1]中首先提出了关系数据模型，以后Codd又提出了关系代数和关系演算的概念，提出了函数依赖的概念，1972年提出了关系的第一、第二、第三范式，1974年提出了BC（Boyce-

Codd)范式,为关系数据库系统奠定了理论基础。由于对关系数据库理论的突出贡献,1981 年 Codd 获得了 ACM 的最高奖——图灵奖。)

2 Codd E F. A Data Base Sublanguage Founded on the Relational Calculus. Proceedings of ACM SIGFIDET Workshop on Data Description. Access and Control,1971

3 Codd E F. Relational Completeness of Data Base Sublanguages in Data Base Systems. Courant Computer Science Symposia Series, Vol. 6,1972

4 Codd E F. Further Normalization of the Data Base Relational Model. In Data Base Systems, Prentice-Hall,1972

5 Codd E F. Recent Investigations in Relational Database Systems. in Proceedings of the IFIP Congess,1974

6 Ullman J, Principles of Database Systems, Second Edition, Computer Science Press,1982

(Ullman 在文献[6]中给出了关系代数、元组关系演算和域关系演算的等价性证明。)

7 Hall P A V, Hitchcock P, Todd S J P. An Algebra of Relations for Machine Computation. Conference Record of the Second ACM Symposium on Principles of Programming Languages,1975

8 Furtado A L, Kerschberg L. An Algebra of Quotient Relations. Proceedings of ACM SIGMOD International Conference on Management of Data,1977

9 Zloof M. M. Query By Example. Proc. NCC, AFIPS, 44, May 1975

(域关系演算的思想出现在文献[9]描述的 QBE 语言中,形式化定义由 Lacroix 和 Pirotte 在文献[10]中给出。文献[11]给出了一个域关系演算语言 ILL。)

10 Lacroix M, Pirotte A. Domain-Oriented Relational Languages. Proc. of 3rd International Conference on Very Large Data Bases,1977

11 Lacroix M, Pirotte A. JLL: An English Structured Query Language for Relational Data Bases. in Nijssen,1977

12 Stonebraker M, Wong E, Kreps P, et al. The Design and Implementation of INGRES, TODS, 1:3,1976

(文献[12]讨论的 QUEL 语言是建立在元组关系演算基础上的查询语言。QUEL 语言是 INGRES 数据库系统的原始查询语言,文献[13]描述了大学版 INGRES 系统的 QUEL 语言,文献[14]介绍了商品化 INGRES 数据库系统的 QUEL 语言。)

13 Zook W, et al. INGRES Reference Manual. Department of EECS, University of California at Berkeley,1977

14 Zook W, et al. RTI INGRES Reference Manual. Relational Technology Inc. ,1983

15 Date C J. Referential Integrity. Proc. of 7th International Conference on Very Large Data Bases,1981

16 Date C J. Why Relations Should Have Exactly One Primary Key. ANSI Database Committee(X3H2) Working Paper X3H2-84-118,1984

第三章 关系数据库标准语言 SQL

SQL (Structured Query Language),即结构化查询语言,是关系数据库的标准语言, SQL 是一个通用的、功能极强的关系数据库语言。其功能并不仅仅是查询。当前,几乎所有的关系数据库管理系统软件都支持 SQL,许多软件厂商对 SQL 基本命令集还进行了不同程度的扩充和修改。

这一章详细介绍 SQL,并进一步讲述关系数据库的基本概念。

3.1 SQL 概述

自 SQL 成为国际标准语言以后,各个数据库厂家纷纷推出各自的 SQL 软件或与 SQL 的接口软件。这就使大多数数据库均用 SQL 作为共同的数据存取语言和标准接口,使不同数据库系统之间的互操作有了共同的基础。SQL 已成为数据库领域中的主流语言。这个意义十分重大。有人把确立 SQL 为关系数据库语言标准及其后的发展称为是一场革命。

3.1.1 SQL 的产生与发展

SQL 是在 1974 年由 Boyce 和 Chamberlin 提出的,并在 IBM 公司研制的关系数据库管理系统原型 System R 上实现。由于 SQL 简单易学,功能丰富,深受用户及计算机工业界欢迎,因此被数据库厂商所采用。经各公司的不断修改、扩充和完善,SQL 得到业界的认可。1986 年 10 月美国国家标准局(American National Standard Institute,简称 ANSI)的数据库委员会 X3H2 批准了 SQL 作为关系数据库语言的美国标准。同年公布了 SQL 标准文本(简称 SQL-86)。1987 年国际标准化组织(International Organization for Standardization,简称 ISO)也通过了这一标准。

SQL 标准从 1986 年公布以来随着数据库技术的发展不断发展,不断丰富。以下是 SQL 标准的进展过程。

标准	大致页数	发布日期
■ SQL/86		1986 年 10 月
■ SQL/89(FIPS 127-1)	120 页	1989 年
■ SQL/92	622 页	1992 年
■ SQL99	1700 页	1999 年
■ SQL2003	360 页	2003 年

可以发现,SQL 标准的内容越来越多,SQL99 合计超过 1700 页。SQL/86 和 SQL/89 都是单个的文档。SQL/92 和 SQL99 已经扩展为一系列开放的部分。例如,SQL/92 除了 SQL 基本部分外还增加了 SQL 调用接口、SQL 永久存储模块。而 SQL99 则进一步扩展为框架、SQL 基础部分、SQL 调用接口、SQL 永久存储模块、SQL 宿主语言绑定、SQL 外部数据的管理和 SQL 对象语言绑定等多个部分。

在《数据库系统概论学习指导与习题解答》[①]的附录 C 中概要介绍了 SQL99 的新特征。

3.1.2 SQL 的特点

SQL 之所以能够为用户和业界所接受,并成为国际标准,是因为它是一个综合的、功能极强同时又简洁易学的语言。SQL 集数据查询(Data Query)、数据操纵(Data Manipulation)、数据定义(Data Definition)和数据控制(Data Control)功能于一体,主要特点包括:

一、综合统一

数据库系统的主要功能是通过数据库支持的数据语言来实现的。

非关系模型(层次模型、网状模型)的数据语言一般都分为:

- 模式数据定义语言(Schema Data Definition Language,模式 DDL);
- 外模式数据定义语言(Subschema Data Definition Language,外模式 DDL 或子模式 DDL);
- 数据存储有关的描述语言(Data Storage Description Language,DSDL);
- 数据操纵语言(Data Manipulation Language,DML)。

它们分别用于定义模式、外模式、内模式和进行数据的存取与处置。当用户数据库投入运行后,如果需要修改模式,必须停止现有数据库的运行,转储数据,修改模式并编译后再重装数据库,十分麻烦。

SQL 则集数据定义语言 DDL、数据操纵语言 DML、数据控制语言 DCL 的功能于一体,语言风格统一,可以独立完成数据库生命周期中的全部活动,包括:

- 定义关系模式,插入数据,建立数据库;
- 对数据库中的数据进行查询和更新;
- 数据库重构和维护;

① 请参见本章参考文献[15]。

• 数据库安全性、完整性控制；

等一系列操作要求。

这就为数据库应用系统的开发提供了良好的环境。特别是用户在数据库系统投入运行后，还可根据需要随时地逐步地修改模式，并不影响数据库的运行，从而使系统具有良好的可扩展性。

另外，在关系模型中实体和实体间的联系均用关系表示，这种数据结构的单一性带来了数据操作符的统一性，查找、插入、删除、更新等每一种操作都只需一种操作符，从而克服了非关系系统由于信息表示方式的多样性带来的操作复杂性。例如，在 DBTG 中，需要两种插入操作符：STORE 用来把记录存入数据库，CONNECT 用来把记录插入系值（系值是网状数据库中记录之间的一种联系方式）以建立数据之间的联系。

二、高度非过程化

非关系数据模型的数据操纵语言是“面向过程”的语言，用“过程化”语言完成某项请求，必须指定存取路径。而用 SQL 进行数据操作，只要提出“做什么”，而无须指明“怎么做”，因此无需了解存取路径。存取路径的选择以及 SQL 的操作过程由系统自动完成。这不但大大减轻了用户负担，而且有利于提高数据独立性。

三、面向集合的操作方式

非关系数据模型采用的是面向记录的操作方式，操作对象是一条记录。例如查询所有平均成绩在 80 分以上的学生姓名，用户必须一条一条地把满足条件的学生记录找出来（通常要说明具体处理过程，即按照哪条路径，如何循环等）。而 SQL 采用集合操作方式，不仅操作对象、查找结果可以是元组的集合，而且一次插入、删除、更新操作的对象也可以是元组的集合。

四、以同一种语法结构提供多种使用方式

SQL 既是独立的语言，又是嵌入式语言。

作为独立的语言，它能够独立地用于联机交互的使用方式，用户可以在终端键盘上直接键入 SQL 命令对数据库进行操作；作为嵌入式语言，SQL 语句能够嵌入到高级语言（例如 C，C++，Java）程序中，供程序员设计程序时使用。而在两种不同的使用方式下，SQL 的语法结构基本上是一致的。这种以统一的语法结构提供多种不同使用方式的做法，提供了极大的灵活性与方便性。

五、语言简洁，易学易用

SQL 功能极强，但由于设计巧妙，语言十分简洁，完成核心功能只用了 9 个动词，如表 3.1 所示。SQL 接近英语口语，因此容易学习，容易使用。

表 3.1 SQL 的动词

SQL 功能	动 词
数据查询	SELECT
数据定义	CREATE,DROP,ALTER
数据操纵	INSERT,UPDATE,DELETE
数据控制	GRANT,REVOKE

3.1.3 SQL 的基本概念

支持 SQL 的 RDBMS 同样支持关系数据库三级模式结构,如图 3.1 所示。其中外模式对应于视图(View)和部分基本表(Base Table),模式对应于基本表,内模式对应于存储文件(Stored File)。

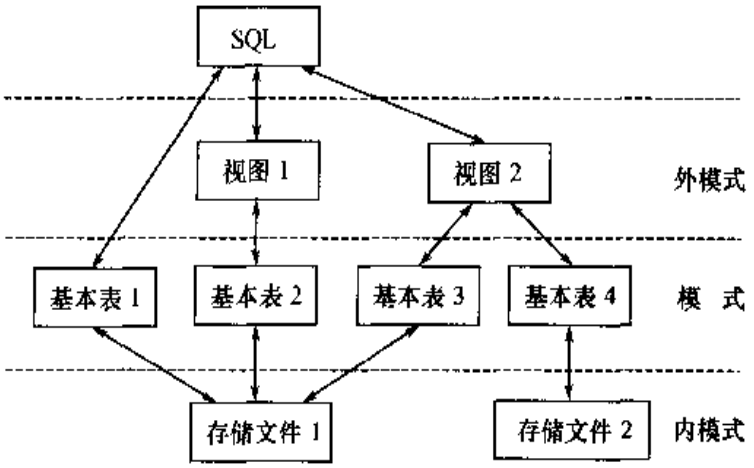


图 3.1 SQL 对关系数据库模式的支持

用户可以用 SQL 对基本表和视图进行查询或其他操作,基本表和视图一样,都是关系。

基本表是本身独立存在的表,在 SQL 中一个关系就对应一个基本表。一个(或多个)基本表对应一个存储文件,一个表可以带若干索引,索引也存放在存储文件中。

存储文件的逻辑结构组成了关系数据库的内模式。存储文件的物理结构是任意的,对用户是透明的。

视图是从一个或几个基本表导出的表。它本身不独立存储在数据库中,即数据库中只存放视图的定义而不存放视图对应的数据。这些数据仍存放在导出视图的基本表中,因此视图是一个虚表。视图在概念上与基本表等同,用户可以在视图上再定义视图。

下面将逐一介绍各 SQL 语句的功能和格式。为了突出基本概念和基本功能,略去了许多语法细节。各个 RDBMS 产品在实现标准 SQL 时各有差别,与 SQL 标准的符合程度也不相同,一般在 85% 以上。因此,具体使用某个 RDBMS 产品时,还应参阅系统提供的有关手册。

3.2　学生－课程数据库

在本章中用学生－课程数据库作为一个例子来讲解 SQL 的数据定义、数据操纵、数据查询和数据控制语句的具体应用。

为此，首先要定义一个学生－课程模式 S-T(见 3.3.1 中[**例 1**])。学生－课程数据库中包括以下 3 个表，它们的定义见 3.3.1 中[**例 5**]、[**例 6**]、[**例 7**]。

- 学生表：Student(Sno，Sname，Ssex，Sage，Sdept)
- 课程表：Course(Cno，Cname，Cpno，Ccredit)
- 学生选课表：SC(Sno，Cno，Grade)

关系的主码加下划线表示。各个表中的数据示例见图 3.2。

Student

学　号 Sno	姓　名 Sname	性　别 Ssex	年　龄 Sage	所在系 Sdept
200215121	李勇	男	20	CS
200215122	刘晨	女	19	CS
200215123	王敏	女	18	MA
200215125	张立	男	19	IS

(a)

Course

课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL 语言	6	4

(b)

SC

学　号 Sno	课 程 号 Cno	成　绩 Grade
200215121	1	92
200215121	2	85
200215121	3	88
200215122	2	90
200215122	3	80

(c)

图 3.2　学生－课程数据库的数据示例

3.3 数据定义

关系数据库系统支持三级模式结构，其模式、外模式和内模式中的基本对象有表、视图和索引。因此 SQL 的数据定义功能包括模式定义、表定义、视图和索引的定义，如表 3.2 所示。

表 3.2 SQL 的数据定义语句

操作对象	操作方式		
	创建	删除	修改
模式	CREATE SCHEMA	DROP SCHEMA	
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	
索引	CREATE INDEX	DROP INDEX	

SQL 通常不提供修改模式定义、修改视图定义和修改索引定义的操作。用户如果想修改这些对象，只能先将它们删除掉，然后再重建。

本节介绍如何定义模式、基本表和索引，视图的概念及其定义方法将在 3.6 节专门讨论。

3.3.1 模式的定义与删除

一、定义模式

在 SQL 中，模式定义语句如下：

CREATE SCHEMA <模式名> AUTHORIZATION <用户名>

如果没有指定 <模式名>，那么 <模式名> 隐含为 <用户名>。

要创建模式，调用该命令的用户必须拥有 DBA 权限，或者获得了 DBA 授予的 CREATE SCHEMA 的权限。

[例 1] 定义一个学生 - 课程模式 S-T

```
CREATE SCHEMA"S-T" AUTHORIZATION WANG;
```

为用户 WANG 定义了一个模式 S-T。

[例 2]

```
CREATE SCIIEMA AUTHORIZATION WANG;
```

该语句没有指定 <模式名>，所以 <模式名> 隐含为用户名 WANG。

定义模式实际上定义了一个命名空间，在这个空间中可以进一步定义该模式包含的数据库对象，例如基本表、视图、索引等。

这些数据库对象可以用表 3.2 中相应的 CREATE 语句来定义。

目前,在 CREATE SCHEMA 中可以接受 CREATE TABLE,CREATE VIEW 和 GRANT 子句。也就是说用户可以在创建模式的同时在这个模式定义中进一步创建基本表、视图,定义授权。即

CREATE SCHEMA <模式名> AUTHORIZATION <用户名> [<表定义子句> | <视图定义子句> | <授权定义子句>]

[例 3]

```
CREATE SCHEMA TEST AUTHORIZATION ZHANG
    CREATE TABLE TAB1(COL1 SMALLINT,
                      COL2 INT,
                      COL3 CHAR(20),
                      COL4 NUMERIC(10,3),
                      COL5 DECIMAL(5,2)
                      );
```

该语句为用户 ZHANG 创建了一个模式 TEST,并且在其中定义了一个表 TAB1。

二、删除模式

在 SQL 中,删除模式语句如下:

DROP SCHEMA <模式名> <CASCADE | RESTRICT>

其中 CASCADE 和 RESTRICT 两者必选其一。

选择了 CASCADE(级联),表示在删除模式的同时把该模式中所有的数据库对象全部一起删除。

选择了 RESTRICT(限制),表示如果该模式中已经定义了下属的数据库对象(如表、视图等),则拒绝该删除语句的执行。只有当该模式中没有任何下属的对象时才能执行 DROP SCHEMA 语句。

[例 4] DROP SCHEMA ZHANG CASCADE;

该语句删除了模式 ZHANG。同时,该模式中已经定义的表 TAB1 也被删除了。

3.3.2 基本表的定义、删除与修改

一、定义基本表

创建了一个模式,就建立了一个数据库的命名空间,一个框架。在这个空间中首先要定义的是该模式包含的数据库基本表。

SQL 语言使用 CREATE TABLE 语句定义基本表,其基本格式如下:

```
CREATE TABLE <表名>( <列名> <数据类型>[列级完整性约束条件]
                    [,<列名> <数据类型>[列级完整性约束条件]]
                    ...
```

[,<表级完整性约束条件>]);

建表的同时通常还可以定义与该表有关的完整性约束条件,这些完整性约束条件被存入系统的数据字典中,当用户操作表中数据时由 RDBMS 自动检查该操作是否违背这些完整性约束条件。如果完整性约束条件涉及到该表的多个属性列,则必须定义在表级上,否则既可以定义在列级也可以定义在表级。

[例 5] 建立一个"学生"表 Student。

```
CREATE TABLE Student
  (Sno CHAR(9) PRIMARY KEY,                    /* 列级完整性约束条件, Sno 是主码 */
   Sname CHAR(20) UNIQUE,                      /* Sname 取唯一值 */
   Ssex CHAR(2),
   Sage SMALLINT,
   Sdept CHAR(20)
  );
```

系统执行上面的 CREATE TABLE 语句后,就在数据库中建立一个新的空的"学生"表 Student,并将有关"学生"表的定义及有关约束条件存放在数据字典中。

[例 6] 建立一个"课程"表 Course。

```
CREATE TABLE Course
  (Cno CHAR(4) PRIMARY KEY,                    /* 列级完整性约束条件,Cno 是主码 */
   Cname CHAR(40),
   Cpno CHAR(4),                               /* Cpno 的含义是先修课 */
   Ccredit SMALLINT,
   FOREIGN KEY Cpno REFERENCES Course(Cno)
  /* 表级完整性约束条件,Cpno 是外码,被参照表是 Course,被参照列是 Cno */
  );
```

本例说明参照表和被参照表可以是同一个表。

[例 7] 建立学生选课表 SC。

```
CREATE TABLE SC
  (Sno CHAR(7),
   Cno CHAR(4),
   Grade SMALLINT,
   PRIMARY KEY (Sno,Cno),
              /* 主码由两个属性构成,必须作为表级完整性进行定义 */
   FOREIGN KEY (Sno) REFERENCES Student(Sno),
              /* 表级完整性约束条件,Sno 是外码,被参照表是 Student */
   FOREIGN KEY (Cno) REFERENCES Course(Cno)
              /* 表级完整性约束条件, Cno 是外码,被参照表是 Course */
  );
```

二、数据类型

关系模型中一个很重要的概念是域。每一个属性来自一个域，它的取值必须是域中的值。

在 SQL 中域的概念用数据类型来实现。定义表的各个属性时需要指明其数据类型及长度。SQL 提供了一些主要数据类型，如表 3.3 所示。要注意，不同的 RDBMS 中支持的数据类型不完全相同。

一个属性选用哪种数据类型要根据实际情况来决定，一般要从两个方面来考虑，一是取值范围，二是要做哪些运算。例如，对于年龄(Sage)属性，可以采用 CHAR(3)作为数据类型，但考虑到要在年龄上做算术运算(如求平均年龄)，所以要采用整数作为数据类型，因为 CHAR(n)数据类型上不能进行算术运算。整数又有长整数和短整数两种，因为一个人的年龄在百岁左右，所以选用短整数作为年龄的数据类型。

表 3.3　数据类型

数据类型	含义
CHAR(n)	长度为 n 的定长字符串
VARCHAR(n)	最大长度为 n 的变长字符串
INT	长整数(也可以写作 INTEGER)
SMALLINT	短整数
NUMERIC(p,d)	定点数，由 p 位数字(不包括符号、小数点)组成，小数后面有 d 位数字
REAL	取决于机器精度的浮点数
Double Precision	取决于机器精度的双精度浮点数
FLOAT(n)	浮点数，精度至少为 n 位数字
DATE	日期，包含年、月、日，格式为 YYYY-MM-DD
TIME	时间，包含一日的时、分、秒，格式为 HH:MM:SS

三、模式与表

每一个基本表都属于某一个模式，一个模式包含多个基本表。当定义基本表时如何定义它所属的模式呢？一般可以有三种方法。例如在[**例 1**]中定义了一个学生－课程模式 S-T。现在要在 S-T 中定义 Student、Cource、SC 等基本表。

方法一，在表名中明显地给出模式名

```
Create table "S-T".Student(.....);          /* Student 所属的模式是 S-T */
Create table "S-T".Cource(.....);           /* Cource 所属的模式是 S-T */
Create table "S-T".SC(.....);               /* SC 所属的模式是 S-T */
```

方法二，在创建模式语句中同时创建表，如[**例 3**]所示。

方法三，设置所属的模式，这样在创建表时表名中不必给出模式名。

当用户创建基本表(其他数据库对象也一样)时若没有指定模式,系统根据搜索路径(search path)来确定该对象所属的模式。

搜索路径包含一组模式列表,RDBMS 会使用模式列表中第一个存在的模式作为数据库对象的模式名。若搜索路径中的模式名都不存在,系统将给出错误。

使用下面的语句可以显示当前的搜索路径:

```
SHOW search_path;
```

搜索路径的当前默认值是:$user, PUBLIC。其含义是首先搜索与用户名相同的模式名,如果该模式名不存在,则使用 PUBLIC 模式。

DBA 用户也可以设置搜索路径,例如:

```
SET search_path TO "S-T",PUBLIC;
```

然后,定义基本表:

```
Create table Student(.....);
```

实际结果是建立了 S-T. Student 基本表。因为 RDBMS 发现搜索路径中第一个模式名 S-T 存在,就把该模式作为基本表 Student 所属的模式。

四、修改基本表

随着应用环境和应用需求的变化,有时需要修改已建立好的基本表,SQL 语言用 ALTER TABLE 语句修改基本表,其一般格式为:

```
ALTER TABLE <表名>
[ ADD <新列名> <数据类型> [ 完整性约束 ]]
[ DROP <完整性约束名> ]
[ ALTER COLUMN <列名> <数据类型> ];
```

其中<表名>是要修改的基本表,ADD 子句用于增加新列和新的完整性约束条件,DROP 子句用于删除指定的完整性约束条件,ALTER COLUMN 子句用于修改原有的列定义,包括修改列名和数据类型。

[例 8] 向 Student 表增加"入学时间"列,其数据类型为日期型。

```
ALTER TABLE Student ADD S_entrance DATE;
```

不论基本表中原来是否已有数据,新增加的列一律为空值。

[例 9] 将年龄的数据类型由字符型(假设原来的数据类型是字符型)改为整数。

```
ALTER TABLE Student ALTER COLUMN Sage INT;
```

[例 10] 增加课程名称必须取唯一值的约束条件。

```
ALTER TABLE Course ADD UNIQUE(Cname);
```

五、删除基本表

当某个基本表不再需要时,可以使用 DROP TABLE 语句删除它。其一般格式为:

```
DROP TABLE <表名> [RESTRICT | CASCADE];
```

若选择 RESTRICT:则该表的删除是有限制条件的。欲删除的基本表不能被其他表的约束所引用(如 CHECK, FOREIGN KEY 等约束),不能有视图,不能有触发器,不能有存储过程或函数等。如果存在这些依赖该表的对象,则此表不能被删除。

若选择 CASCADE:则该表的删除没有限制条件。在删除基本表的同时,相关的依赖对象,例如视图,都将被一起删除。

缺省情况是 RESTRICT。

[例 11] 删除 Student 表。

```
DROP TABLE Student CASCADE;
```

基本表定义一旦被删除,不仅表中的数据和此表的定义将被删除,而且此表上建立的索引、视图、触发器等有关对象一般也都将被删除。因此执行删除基本表的操作一定要格外小心。

[例 12] 若表上建有视图,选择 RESTRICT 时表不能删除;CASCADE 时可以删除表,视图也自动被删除。

```
CREATE VIEW IS_Student                    /* Student 表上建立视图 */
AS
SELECT Sno,Sname,Sage
FROM Student
WHERE Sdept ='IS';

DROP TABLE Student RESTRICT;              /* 删除 Student 表 */
-- ERROR: cannot drop table Student because other objects depend on it
                                          /* 系统返回错误信息,存在依赖该表的对象,此表不能被删除 */
DROP TABLE Student CASCADE;               /* 删除 Student 表 */
-- NOTICE: drop cascades to view IS_Student
                                          /* 系统返回提示,此表上的视图也被删除 */
SELECT * FROM IS_Student;
-- ERROR: relation " IS_Student " does not exist
```

注意:不同的数据库产品在遵循 SQL 标准的基础上具体实现细节和处理策略上会与标准有差别。

下面就 SQL99 标准对 DROP TABLE 的规定,对比分析 Kingbase ES、ORACLE 9i、MS SQLSERVER 2000 这 3 种数据库产品对 DROP TABLE 的不同处理策略。

表中的 R 表示 RESTRICT,即 DROP TABLE <基本表名> RESTRICT; C 表示 CASCADE,即 DROP TABLE <基本表名> CASCADE;其中 Oracle9i 没有 RESTRICT 选项;SQL SERVER 没有 RESTRICT 和 CASCADE 选项。

DROP TABLE 时,SQL99 与 3 个 RDBMS 的处理策略比较

序号	标准及主流数据库的处理方式 / 依赖基本表的对象	SQL99		Kingbase ES		ORACLE 9i		MS SQL SERVER 2000
		R	C	R	C		C	
1.	索引	无规定		✓	✓	✓	✓	✓
2.	视图	×	✓	×	✓	✓ 保留	✓ 保留	✓ 保留
3.	DEFAULT,PRIMARY KEY,CHECK(只含该表的列)NOT NULL 等约束	✓	✓	✓	✓	✓	✓	✓
4.	Foreign Key	×	✓	×	✓	×	✓	×
5.	TRIGGER	×	✓	×	✓	✓	✓	✓
6.	函数或存储过程	×	✓	✓ 保留	✓ 保留	✓ 保留	✓ 保留	✓ 保留

'×'表示不能删除基本表,'✓'表示能删除基本表,'保留'表示删除基本表后,还保留依赖对象。从比较表中可以知道:

1. 对于索引,删除基本表后,这 3 个 RDBMS 都自动删除该基本表上已经建立的所有索引;

2. 对于视图,ORACLE 9i 与 SQL SERVER 2000 是删除基本表后,还保留此基本表上的视图定义,但是已经失效。KingbaseES 分两种情况,若删除基本表时带 RESTRICT 选项,则不可以删除基本表;若删除基本表时带 CASCADE 选项,可以删除基本表,同时也删除视图;KingbaseES 的这种策略符合 SQL99 标准。

3. 对于存储过程和函数,删除基本表后,这 3 个数据库产品都不自动删除建立在此基本表上的存储过程和函数,但是已经失效。

4. 如果欲删除的基本表上有触发器,或者被其他基本表的约束所引用(CHECK,FOREIGN KEY 等),读者可以从比较表中得到这 3 个系统的处理策略,这里就不一一说明了。

同样,对于其他的 SQL 语句,不同的数据库产品在处理策略上会与标准有所差别。因此,如果发现本书中个别例子在某个数据库产品上不能通过时,请读者参见有关产品的用户手册,适当修改即可。

3.3.3 索引的建立与删除

建立索引是加快查询速度的有效手段。用户可以根据应用环境的需要,在基本表上建立一个或多个索引,以提供多种存取路径,加快查找速度。

一般说来,建立与删除索引由数据库管理员 DBA 或表的属主(owner),即建立表的人,负责完成。系统在存取数据时会自动选择合适的索引作为存取路径,用户不必也不能显式地选择索引。

一、建立索引

在 SQL 语言中，建立索引使用 CREATE INDEX 语句，其一般格式为：

```
CREATE [ UNIQUE ][ CLUSTER ]INDEX <索引名>
ON <表名>(<列名> [ <次序>][,<列名> [ <次序>]]…);
```

其中，<表名>是要建索引的基本表的名字。索引可以建立在该表的一列或多列上，各列名之间用逗号分隔。每个<列名>后面还可以用<次序>指定索引值的排列次序，可选 ASC（升序）或 DESC（降序），缺省值为 ASC。

UNIQUE 表明此索引的每一个索引值只对应唯一的数据记录。

CLUSTER 表示要建立的索引是聚簇索引。所谓聚簇索引是指索引项的顺序与表中记录的物理顺序一致的索引组织。例如，执行下面的 CREATE INDEX 语句：

[例 13] CREATE CLUSTER INDEX Stusname ON Student(Sname);

将会在 Student 表的 Sname（姓名）列上建立一个聚簇索引，而且 Student 表中的记录将按照 Sname 值的升序存放。

用户可以在最经常查询的列上建立聚簇索引以提高查询效率。显然在一个基本表上最多只能建立一个聚簇索引。建立聚簇索引后，更新该索引列上的数据时，往往导致表中记录的物理顺序的变更，代价较大，因此对于经常更新的列不宜建立聚簇索引。

[例 14] 为学生－课程数据库中的 Student，Course，SC 3 个表建立索引。其中 Student 表按学号升序建唯一索引，Course 表按课程号升序建唯一索引，SC 表按学号升序和课程号降序建唯一索引。

```
CREATE UNIQUE INDEX Stusno ON Student(Sno);
CREATE UNIQUE INDEX Coucno ON Course(Cno);
CREATE UNIQUE INDEX SCno ON SC(Sno ASC,Cno DESC);
```

二、删除索引

索引一经建立，就由系统使用和维护它，不需用户干预。建立索引是为了减少查询操作的时间，但如果数据增删改频繁，系统会花费许多时间来维护索引，从而降低了查询效率。这时，可以删除一些不必要的索引。

在 SQL 中，删除索引使用 DROP INDEX 语句，其一般格式为

```
DROP INDEX <索引名>;
```

[例 15] 删除 Student 表的 Stusname 索引。

```
DROP INDEX Stusname;
```

删除索引时，系统会同时从数据字典中删去有关该索引的描述。

在 RDBMS 中索引一般采用 B＋树、HASH 索引来实现。B＋树索引具有动态平衡的优点。HASH 索引具有查找速度快的特点。索引是关系数据库的内部实现技术，属于内模式的范畴。

用户使用 CREATE INDEX 语句定义索引时，可以定义索引是唯一索引、非唯一索引或聚簇索引。至于某一个索引是采用 B + 树，还是 HASH 索引则由具体的 RDBMS 来决定。

3.4 数据查询

3.3 讲解了 SQL 的模式定义、基本表定义和索引的定义。视图的定义和概念放在 3.6 中讲解。这一节讲解 SQL 查询功能。数据库查询是数据库的核心操作。SQL 提供了 SELECT 语句进行数据库的查询，该语句具有灵活的使用方式和丰富的功能。其一般格式为：

```
SELECT [ALL | DISTINCT] <目标列表达式> [,<目标列表达式>]…
FROM <表名或视图名> [, <表名或视图名> ]…
[ WHERE <条件表达式> ]
[ GROUP BY <列名 1> [ HAVING <条件表达式> ]]
[ ORDER BY <列名 2> [ ASC | DESC ]];
```

整个 SELECT 语句的含义是，根据 WHERE 子句的条件表达式，从 FROM 子句指定的基本表或视图中找出满足条件的元组，再按 SELECT 子句中的目标列表达式，选出元组中的属性值形成结果表。

如果有 GROUP BY 子句，则将结果按 <列名 1> 的值进行分组，该属性列值相等的元组为一个组。通常会在每组中作用聚集函数（aggregate functions）。如果 GROUP BY 子句带 HAVING 短语，则只有满足指定条件的组才予以输出。

如果有 ORDER BY 子句，则结果表还要按 <列名 2> 的值的升序或降序排序。

SELECT 语句既可以完成简单的单表查询，也可以完成复杂的连接查询和嵌套查询。下面以学生 - 课程数据库为例说明 SELECT 语句的各种用法。

3.4.1 单表查询

单表查询是指仅涉及一个表的查询。

一、选择表中的若干列

选择表中的全部列或部分列，这就是关系代数的投影运算。

1. 查询指定列

在很多情况下，用户只对表中的一部分属性列感兴趣，这时可以通过在 SELECT 子句的 <目标列表达式> 中指定要查询的属性列。

[例 1] 查询全体学生的学号与姓名。

```
SELECT Sno,Sname
FROM Student;
```

该语句的执行过程可以是这样的：从 Student 表中取出一个元组，取出该元组在属性 Sno 和 Sname 上的值，形成一个新的元组作为输出。对 Student 表中的所有元组做相同的处理，

最后形成一个结果关系作为输出。

[例 2] 查询全体学生的姓名、学号、所在系。

```
SELECT Sname,Sno,Sdept
FROM Student;
```

<目标列表达式>中各个列的先后顺序可以与表中的顺序不一致。用户可以根据应用的需要改变列的显示顺序。本例中先列出姓名,再列出学号和所在系。

2. 查询全部列

将表中的所有属性列都选出来,可以有两种方法。一种方法就是在 SELECT 关键字后面列出所有列名。如果列的显示顺序与其在基表中的顺序相同,也可以简单地将<目标列表达式>指定为 *。

[例 3] 查询全体学生的详细记录。

```
SELECT *
FROM Student;
```

等价于:

```
SELECT Sno,Sname,Ssex,Sage,Sdept
FROM Student;
```

3. 查询经过计算的值

SELECT 子句的<目标列表达式>不仅可以是表中的属性列,也可以是表达式。

[例 4] 查询全体学生的姓名及其出生年份。

```
SELECT Sname,2004 - Sage        /* 查询结果的第 2 列是一个算术表达式 */
FROM Student;
```

查询结果中第 2 列不是列名而是一个计算表达式,是用当时的年份(假设为 2004 年)减去学生的年龄。这样,所得的即是学生的出生年份。输出的结果为:

Sname	2004 - Sage
李勇	1984
刘晨	1985
王敏	1986
张立	1985

<目标列表达式>不仅可以是算术表达式,还可以是字符串常量、函数等。

[例 5] 查询全体学生的姓名、出生年份和所在的院系,要求用小写字母表示所有系名。

```
SELECT Sname,' Year of Birth: ',2004 - Sage,LOWER(Sdept)
FROM Student;
```

结果为:

Sname	'Year of Birth:'	2004 - Sage	LOWER(Sdept)
李勇	Year of Birth:	1984	cs
刘晨	Year of Birth:	1985	cs
王敏	Year of Birth:	1986	ma
张立	Year of Birth:	1985	is

用户可以通过指定别名来改变查询结果的列标题,这对于含算术表达式、常量、函数名的目标列表达式尤为有用。例如对于上例,可以定义如下列别名:

```
SELECT Sname NAME,'Year of Birth:' BIRTH,2004 - Sage BIRTHDAY,
       LOWER(Sdept) DEPARTMENT
FROM Student;
```

结果为:

NAME	BIRTH	BIRTHDAY	DEPARTMENT
李勇	Year of Birth:	1984	cs
刘晨	Year of Birth:	1985	cs
王敏	Year of Birth:	1986	ma
张立	Year of Birth:	1985	is

二、选择表中的若干元组

1. 消除取值重复的行

两个本来并不完全相同的元组,投影到指定的某些列上后,可能变成相同的行了,可以用 DISTINCT 取消它们。

[例 6] 查询选修了课程的学生学号。

```
SELECT Sno
FROM SC;
```

执行上面的 SELECT 语句后,结果为:

Sno
200215121
200215121
200215121
200215122
200215122

该查询结果里包含了许多重复的行。如果想去掉结果表中的重复行,必须指定 DISTINCT 关键词:

```
SELECT DISTINCT Sno
FROM SC;
```

执行结果为:

Sno
200215121
200215122

如果没有指定 DISTINCT 关键词,则缺省为 ALL,即保留结果表中取值重复的行。

```
SELECT Sno
FROM SC;
```

等价于:

```
SELECT ALL Sno
FROM SC;
```

2. 查询满足条件的元组

查询满足指定条件的元组可以通过 WHERE 子句实现。WHERE 子句常用的查询条件如表 3.4 所示。

表 3.4 常用的查询条件

查询条件	谓 词
比较	=,>,<,>=,<=,!=,<>,!>,!<;NOT + 上述比较运算符
确定范围	BETWEEN AND,NOT BETWEEN AND
确定集合	IN,NOT IN
字符匹配	LIKE,NOT LIKE
空值	IS NULL,IS NOT NULL
多重条件(逻辑运算)	AND,OR,NOT

(1) 比较大小

用于进行比较的运算符一般包括:

=(等于),>(大于),<(小于),>=(大于等于),<=(小于等于),!= 或 <>(不等于),!>(不大于),!<(不小于)。

[例 7] 查询计算机科学系全体学生的名单。

```
SELECT Sname
FROM Student
WHERE Sdept = 'CS';
```

RDBMS 执行该查询的一种可能过程是:对 Student 表进行全表扫描,取出一个元组,检查该元组在 Sdept 列的值是否等于'CS'。如果相等,则取出 Sname 列的值形成一个新的元组输出,否则跳过该元组,取下一个元组。

如果全校有数万个学生,计算机系的学生人数是全校学生的 5% 左右,可以在 Student 表的 Sdept 列上建立索引,系统会利用该索引找出 Sdept = 'CS'的元组,从中取出 Sname 列值形

成结果关系。这就避免了对 Student 表的全表扫描，加快查询速度。注意如果学生较少，索引查找不一定能提高查询效率，系统仍会使用全表扫描。这由查询优化器按照某些规则或估计执行代价来作出选择。

[例 8] 查询所有年龄在 20 岁以下的学生姓名及其年龄。

```
SELECT Sname,Sage
FROM Student
WHERE Sage < 20;
```

[例 9] 查询考试成绩有不及格的学生的学号。

```
SELECT DISTINCT Sno
FROM SC
WHERE Grade < 60;
```

这里使用了 DISTINCT 短语，当一个学生有多门课程不及格，他的学号也只列一次。

(2) 确定范围

谓词 BETWEEN…AND…和 NOT BETWEEN…AND…可以用来查找属性值在(或不在)指定范围内的元组，其中 BETWEEN 后是范围的下限(即低值)，AND 后是范围的上限(即高值)。

[例 10] 查询年龄在 20～23 岁(包括 20 岁和 23 岁)之间的学生的姓名、系别和年龄。

```
SELECT Sname,Sdept,Sage
FROM Student
WHERE Sage BETWEEN 20 AND 23;
```

与 BETWEEN…AND…相对的谓词是 NOT BETWEEN…AND…。

[例 11] 查询年龄不在 20～23 岁之间的学生姓名、系别和年龄。

```
SELECT Sname,Sdept,Sage
FROM Student
WHERE Sage NOT BETWEEN 20 AND 23;
```

(3) 确定集合

谓词 IN 可以用来查找属性值属于指定集合的元组。

[例 12] 查询计算机科学系(CS)、数学系(MA)和信息系(IS)学生的姓名和性别。

```
SELECT Sname,Ssex
FROM Student
WHERE Sdept IN ('CS','MA','IS');
```

与 IN 相对的谓词是 NOT IN，用于查找属性值不属于指定集合的元组。

[例 13] 查询既不是计算机科学系、数学系，也不是信息系的学生的姓名和性别。

```
SELECT Sname,Ssex
FROM Student
WHERE Sdept NOT IN ('CS','MA','IS');
```

(4) 字符匹配

谓词 LIKE 可以用来进行字符串的匹配。其一般语法格式如下:

[NOT]LIKE '<匹配串>'[ESCAPE '<换码字符>']

其含义是查找指定的属性列值与<匹配串>相匹配的元组。<匹配串>可以是一个完整的字符串,也可以含有通配符%和_。其中:

- %(百分号)代表任意长度(长度可以为0)的字符串。例如a%b表示以a开头,以b结尾的任意长度的字符串。如acb,addgb,ab等都满足该匹配串。
- _(下横线)代表任意单个字符。

例如a_b表示以a开头,以b结尾的长度为3的任意字符串。如acb,afb等都满足该匹配串。

[例14] 查询学号为200215121的学生的详细情况。

```
SELECT *
FROM Student
WHERE Sno LIKE '200215121';
```

等价于:

```
SELECT *
FROM Student
WHERE Sno ='200215121';
```

如果LIKE后面的匹配串中不含通配符,则可以用=(等于)运算符取代LIKE谓词,用!=或 <>(不等于)运算符取代NOT LIKE谓词。

[例15] 查询所有姓刘的学生的姓名、学号和性别。

```
SELECT Sname,Sno,Ssex
FROM Student
WHERE Sname LIKE '刘%';
```

[例16] 查询姓"欧阳"且全名为3个汉字的学生的姓名。

```
SELECT Sname
FROM Student
WHERE Sname LIKE '欧阳__';
```

注意:一个汉字要占两个字符的位置,所以匹配串欧阳后面需要跟两个_。

[例17] 查询名字中第2个字为"阳"字的学生的姓名和学号。

```
SELECT Sname,Sno
FROM Student
WHERE Sname LIKE '__阳%';
```

[例 18] 查询所有不姓刘的学生姓名。

```
SELECT Sname,Sno,Ssex
FROM Student
WHERE Sname NOT LIKE '刘%';
```

如果用户要查询的字符串本身就含有通配符 % 或 _,这时就要使用 ESCAPE '<换码字符>' 短语,对通配符进行转义了。

[例 19] 查询 DB_Design 课程的课程号和学分。

```
SELECT Cno,Ccredit
FROM Course
WHERE Cname LIKE 'DB\_Design' ESCAPE '\';
```

ESCAPE '\' 表示“ \” 为换码字符。这样匹配串中紧跟在“ \” 后面的字符“_”不再具有通配符的含义,转义为普通的“_”字符。

[例 20] 查询以“DB_”开头,且倒数第 3 个字符为 i 的课程的详细情况。

```
SELECT *
FROM Course
WHERE Cname LIKE 'DB\_%i_ _' ESCAPE '\';
```

这里的匹配串为“DB_%i_ _”。第一个 _ 前面有换码字符\,所以它被转义为普通的 _ 字符。而 i 后面的两个_ 的前面均没有换码字符\,所以它们仍作为通配符。

(5) 涉及空值的查询

[例 21] 某些学生选修课程后没有参加考试,所以有选课记录,但没有考试成绩。查询缺少成绩的学生的学号和相应的课程号。

```
SELECT Sno,Cno
FROM SC
WHERE Grade IS NULL;                    /* 分数 Grade 是空值 */
```

注意这里的“IS”不能用等号(=)代替。

[例 22] 查所有有成绩的学生学号和课程号。

```
SELECT Sno,Cno
FROM SC
WHERE Grade IS NOT NULL;
```

(6) 多重条件查询

逻辑运算符 AND 和 OR 可用来联结多个查询条件。AND 的优先级高于 OR，但用户可以用括号改变优先级。

[例 23] 查询计算机科学系年龄在 20 岁以下的学生姓名。

```
SELECT Sname
FROM Student
WHERE Sdept = 'CS' AND Sage < 20;
```

在[例 12]中的 IN 谓词实际上是多个 OR 运算符的缩写，因此[例 12]中的查询也可以用 OR 运算符写成如下等价形式：

```
SELECT Sname,Ssex
FROM Student
WHERE Sdept = 'CS' OR Sdept = 'MA' OR Sdept = 'IS';
```

三、ORDER BY 子句

用户可以用 ORDER BY 子句对查询结果按照一个或多个属性列的升序(ASC)或降序(DESC)排列，缺省值为升序。

[例 24] 查询选修了 3 号课程的学生的学号及其成绩，查询结果按分数的降序排列。

```
SELECT Sno,Grade
FROM SC
WHERE Cno = '3'
ORDER BY Grade DESC;
```

对于空值，若按升序排，含空值的元组将最后显示。若按降序排，空值的元组将最先显示。

[例 25] 查询全体学生情况，查询结果按所在系的系号升序排列，同一系中的学生按年龄降序排列。

```
SELECT *
FROM Student
ORDER BY Sdept,Sage DESC;
```

四、聚集函数(aggregate functions)

为了进一步方便用户，增强检索功能，SQL 提供了许多聚集函数，主要有：

COUNT([DISTINCT\|ALL] *)	统计元组个数
COUNT([DISTINCT\|ALL] <列名>)	统计一列中值的个数
SUM([DISTINCT\|ALL] <列名>)	计算一列值的总和(此列必须是数值型)
AVG([DISTINCT\|ALL] <列名>)	计算一列值的平均值(此列必须是数值型)
MAX([DISTINCT\|ALL] <列名>)	求一列值中的最大值
MIN([DISTINCT\|ALL] <列名>)	求一列值中的最小值

如果指定 DISTINCT 短语,则表示在计算时要取消指定列中的重复值。如果不指定 DISTINCT 短语或指定 ALL 短语(ALL 为缺省值),则表示不取消重复值。

[例 26] 查询学生总人数。

```
SELECT COUNT( * )
FROM Student;
```

[例 27] 查询选修了课程的学生人数。

```
SELECT COUNT(DISTINCT Sno)
FROM SC;
```

学生每选修一门课,在 SC 中都有一条相应的记录。一个学生要选修多门课程,为避免重复计算学生人数,必须在 COUNT 函数中用 DISTINCT 短语。

[例 28] 计算 1 号课程的学生平均成绩。

```
SELECT AVG(Grade)
FROM SC
WHERE Cno = '1';
```

[例 29] 查询选修 1 号课程的学生最高分数。

```
SELECT MAX(Grade)
FROM SC
WHER Cno = '1';
```

[例 30] 查询学生 200215012 选修课程的总学分数。

```
SELECT SUM(Ceredit)
FROM SC, Course
WHER Sno = '200215012' AND SC. Cno = Course. Cno;
```

在聚集函数遇到空值时,除 COUNT(*)外,都跳过空值而只处理非空值。

五、GROUP BY 子句

GROUP BY 子句将查询结果按某一列或多列的值分组,值相等的为一组。

对查询结果分组的目的是为了细化聚集函数的作用对象。如果未对查询结果分组,聚集函数将作用于整个查询结果,如上面的[例 26]至[例 30]。分组后聚集函数将作用于每一个组,即每一组都有一个函数值。

[例 31] 求各个课程号及相应的选课人数。

```
SELECT Cno, COUNT(Sno)
FROM SC
GROUP BY Cno;
```

该语句对查询结果按 Cno 的值分组,所有具有相同 Cno 值的元组为一组,然后对每一组作用聚集函数 COUNT 计算,以求得该组的学生人数。

查询结果可能为：

Cno	COUNT(Sno)
1	22
2	34
3	44
4	33
5	48

如果分组后还要求按一定的条件对这些组进行筛选，最终只输出满足指定条件的组，则可以使用 HAVING 短语指定筛选条件。

[例 32] 查询选修了 3 门以上课程的学生学号。

```
SELECT Sno
FROM SC
GROUP BY Sno
HAVING COUNT( * ) >3;
```

这里先用 GROUP BY 子句按 Sno 进行分组，再用聚集函数 COUNT 对每一组计数。HAVING 短语给出了选择组的条件，只有满足条件（即元组个数 >3，表示此学生选修的课超过 3 门）的组才会被选出来。

WHERE 子句与 HAVING 短语的区别在于作用对象不同。WHERE 子句作用于基本表或视图，从中选择满足条件的元组。HAVING 短语作用于组，从中选择满足条件的组。

3.4.2 连接查询

前面的查询都是针对一个表进行的。若一个查询同时涉及两个以上的表，则称之为连接查询。连接查询是关系数据库中最主要的查询，包括等值连接查询、自然连接查询、非等值连接查询、自身连接查询、外连接查询和复合条件连接查询等。

一、等值与非等值连接查询

连接查询的 WHERE 子句中用来连接两个表的条件称为**连接条件**或连接谓词，其一般格式为：

[<表名 1>.] <列名 1> <比较运算符> [<表名 2>.] <列名 2>

其中比较运算符主要有：=、>、<、>=、<=、!=（或 <>）等。

此外连接谓词还可以使用下面形式：

[<表名 1>.] <列名 1> BETWEEN [<表名 2>.] <列名 2> AND [<表名 2>.] <列名 3>

当连接运算符为 = 时，称为**等值连接**。使用其他运算符称为**非等值连接**。

连接谓词中的列名称为**连接字段**。连接条件中的各连接字段类型必须是可比的，但名字不必相同。

[例 33] 查询每个学生及其选修课程的情况。

学生情况存放在 Student 表中，学生选课情况存放在 SC 表中，所以本查询实际上涉及 Student 与 SC 两个表。这两个表之间的联系是通过公共属性 Sno 实现的。

```
SELECT Student.*,SC.*
FROM Student,SC
WHERE Student.Sno = SC.Sno;          /* 将 Student 与 SC 中同一学生的元组连接起来 */
```

假设 Student 表、SC 表的数据如图 3.2 所示，该查询的执行结果如图 3.3 所示：

Student.Sno	Sname	Ssex	Sage	Sdept	SC.Sno	Cno	Grade
200215121	李勇	男	20	CS	200215121	1	92
200215121	李勇	男	20	CS	200215121	2	85
200215121	李勇	男	20	CS	200215121	3	88
200215122	刘晨	女	19	CS	200215122	2	90
200215122	刘晨	女	19	CS	200215122	3	80

图 3.3 [例 33] 查询结果

本例中，SELECT 子句与 WHERE 子句中的属性名前都加上了表名前缀，这是为了避免混淆。如果属性名在参加连接的各表中是唯一的，则可以省略表名前缀。

RDBMS 执行该连接操作的一种可能过程是：

首先在表 Student 中找到第一个元组，然后从头开始扫描 SC 表，逐一查找与 Student 第一个元组的 Sno 相等的 SC 元组，找到后就将 Student 中的第一个元组与该元组拼接起来，形成结果表中一个元组。SC 全部查找完后，再找 Student 中第二个元组，然后再从头开始扫描 SC，逐一查找满足连接条件的元组，找到后就将 Student 中的第二个元组与该元组拼接起来，形成结果表中一个元组。重复上述操作，直到 Student 中的全部元组都处理完毕为止。这就是嵌套循环算法的基本思想。

如果在 SC 表 Sno 上建立了索引的话，就不用每次全表扫描 SC 表了，而是根据 Sno 值通过索引找到相应的 SC 元组。用索引查询 SC 中满足条件的元组一般会比全表扫描快。

若在等值连接中把目标列中重复的属性列去掉则为**自然连接**。

[例 34] 对[例 33]用自然连接完成。

```
SELECT Student.Sno,Sname,Ssex,Sage,Sdept,Cno,Grade
FROM Student,SC
WHERE Student.Sno = SC.Sno;
```

例中，由于 Sname，Ssex，Sage，Sdept，Cno 和 Grade 属性列在 Student 表与 SC 表中是唯一的，因此引用时可以去掉表名前缀。而 Sno 在两个表都出现了，因此引用时必须加上表名前缀。

二、自身连接

连接操作不仅可以在两个表之间进行，也可以是一个表与其自己进行连接，称为表的**自身连接**。

[**例 35**]　查询每一门课的间接先修课(即先修课的先修课)。

在 Course 表中，只有每门课的直接先修课信息，而没有先修课的先修课。要得到这个信息，必须先对一门课找到其先修课，再按此先修课的课程号，查找它的先修课程。这就要将 Course 表与其自身连接。

为此，要为 Course 表取两个别名，一个是 FIRST，另一个是 SECOND。

FIRST 表(Course 表)

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL 语言	6	4

SECOND 表(Course 表)

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL 语言	6	4

完成该查询的 SQL 语句为

```
SELECT FIRST. Cno,SECOND. Cpno
FROM Course FIRST,Course SECOND
WHERE FIRST. Cpno = SECOND. Cno;
```

结果为

Cno	Pcno
1	7
3	5
5	6

三、外连接

在通常的连接操作中，只有满足连接条件的元组才能作为结果输出。如[**例 33**]的结果表中没有 200215123 和 200215125 两个学生的信息，原因在于他们没有选课，在 SC 表中没有相应的元组，造成 Student 中这些元组在连接时被舍弃了。

有时想以 Student 表为主体列出每个学生的基本情况及其选课情况。若某个学生没有选课，仍把舍弃的 Student 元组保存在结果关系中，而在 SC 表的属性上填空值(Null)，这时就需要使用**外连接**，外连接的概念已经在 2.4.2 中讲解过。可以如下改写[**例 33**]：

[**例 36**]

```
SELECT Student. Sno, Sname, Ssex, Sage, Sdept, Cno, Grade
FROM Student LEFT OUT JOIN SC ON (Student. Sno = SC. Sno);
/* 也可以使用 USING 来去掉结果中的重复值:
   FROM Student LEFT OUT JOIN SC USING (Sno); */
```

执行结果如下:

Student. Sno	Sname	Ssex	Sage	Sdept	Cno	Grade
200215121	李勇	男	20	CS	1	92
200215121	李勇	男	20	CS	2	85
200215121	李勇	男	20	CS	3	88
200215122	刘晨	女	19	CS	2	90
200215122	刘晨	女	19	CS	3	80
200215123	王敏	女	18	MA	NULL	NULL
200215125	张立	男	19	IS	NULL	NULL

左外连接列出左边关系(如本例 Student)中所有的元组,**右外连接**列出右边关系中所有的元组。

四、复合条件连接

上面各个连接查询中,WHERE 子句中只有一个条件,即连接谓词。WHERE 子句中可以有多个连接条件,称为**复合条件连接**。

[**例 37**] 查询选修 2 号课程且成绩在 90 分以上的所有学生。

```
SELECT Student. Sno, Sname
FROM Student, SC
WHERE Student. Sno = SC. Sno AND                    /* 连接谓词 */
      SC. Cno = '2' AND SC. Grade > 90;             /* 其他限定条件 */
```

该查询的一种优化(高效)的执行过程是先从 SC 中挑选出 Cno = '2'并且 Grade > 90 的元组形成一个中间关系,再和 Student 中满足连接条件的元组进行连接得到最终的结果关系。

连接操作除了可以是**两表连接**,一个表与其自身连接外,还可以是两个以上的表进行连接,后者通常称为**多表连接**。

[**例 38**] 查询每个学生的学号、姓名、选修的课程名及成绩。

本查询涉及 3 个表,完成该查询的 SQL 语句如下:

```
SELECT Student. Sno, Sname, Cname, Grade
FROM Student, SC, Course
WHERE Student. Sno = SC. Sno and SC. Cno = Course. Cno;
```

3.4.3 嵌套查询

在 SQL 语言中,一个 SELECT-FROM-WHERE 语句称为一个**查询块**。将一个查询块嵌套在另一个查询块的 WHERE 子句或 HAVING 短语的条件中的查询称为**嵌套查询**。例如:

```
SELECT Sname                                    /* 外层查询或父查询 */
FROM Student
WHERE Sno IN
        (SELECT Sno                             /* 内层查询或子查询 */
        FROM SC
        WHERE Cno = '2');
```

本例中,下层查询**块** SELECT Sno FROM SC WHERE Cno = '2'是嵌套在上层查询块 SELECT Sname FROM Student WHERE Sno IN 的 WHERE 条件中的。上层的查询块称为**外层查询**或**父查询**,下层查询块称为**内层查询**或**子查询**。

SQL 语言允许多层嵌套查询。即一个子查询中还可以嵌套其他子查询。需要特别指出的是,子查询的 SELECT 语句中**不能**使用 ORDER BY 子句,ORDER BY 子句只能对最终查询结果排序。

嵌套查询使我们可以用多个简单查询构成复杂的查询,从而增强 SQL 的查询能力。以层层嵌套的方式来构造程序正是 SQL(Structured Query Language)中"结构化"的含义所在。

一、带有 IN 谓词的子查询

在嵌套查询中,子查询的结果往往是一个集合,所以谓词 IN 是嵌套查询中最经常使用的谓词。

[例 39] 查询与"刘晨"在同一个系学习的学生。

先分步来完成此查询,然后再构造嵌套查询。

① 确定"刘晨"所在系名

```
SELECT Sdept
FROM Student
WHERE Sname = '刘晨';
```

结果为:CS

② 查找所有在 CS 系学习的学生。

```
SELECT Sno,Sname,Sdept
FROM Student
WHERE Sdept = 'CS';
```

结果为:

Sno	Sname	Sdept
200215121	李勇	CS
200215122	刘晨	CS

将第一步查询嵌入到第二步查询的条件中，构造嵌套查询如下：

```
SELECT Sno,Sname,Sdept                    /* [例 39]的解法…,后面会讲解其他解法 */
FROM Student
WHERE Sdept IN
      (SELECT Sdept
      FROM Student
      WHERE Sname ='刘晨');
```

本例中，子查询的查询条件不依赖于父查询，称为**不相关子查询**。一种求解方法是由里向外处理，即先执行子查询，子查询的结果用于建立其父查询的查找条件。得到如下的语句：

```
SELECT Sno,Sname,Sdept
FROM Student
WHERE Sdept IN('CS');
```

然后执行该语句。

本例中的查询也可以用自身连接来完成：

```
SELECT S1. Sno,S1. Sname,S1. Sdept          /* [例 39]的解法二 */
FROM Student S1,Student S2
WHERE S1. Sdept = S2. Sdept AND
      S2. Sname ='刘晨';
```

可见，实现同一个查询可以有多种方法，当然不同的方法其执行效率可能会有差别，甚至会差别很大。这就是数据库编程人员应该掌握的数据库性能调优技术，有兴趣的读者可以参考相关文献资料[16]，包括具体产品的性能调优方法。

[例 40] 查询选修了课程名为“信息系统”的学生学号和姓名。

本查询涉及学号、姓名和课程名 3 个属性。学号和姓名存放在 Student 表中，课程名存放在 Course 表中，但 Student 与 Course 两个表之间没有直接联系，必须通过 SC 表建立它们二者之间的联系。所以本查询实际上涉及 3 个关系。

```
SELECT Sno,Sname                          ③ 最后在 Student 关系中
FROM Student                                 取出 Sno 和 Sname
WHERE Sno IN
      (SELECT Sno                         ② 然后在 SC 关系中找出
       FROM SC                               选修了 3 号课程的学生
       WHERE Cno IN                          学号
```

```
        (SELECT Cno                    ① 首先在 Course 关系中
         FROM Course                     找出"信息系统"的课
         WHERE Cname ='信息系统'          程号,结果为 3 号
        )
);
```

本查询同样可以用连接查询实现:

```
SELECT Student.Sno,Sname
FROM Student,SC,Course
WHERE Student.Sno = SC.Sno AND
      SC.Cno = Course.Cno AND
      Course.Cname =' 信息系统 ';
```

从[**例 39**]和[**例 40**]可以看到,查询涉及多个关系时,用嵌套查询逐步求解,层次清楚,易于构造,具有结构化程序设计的优点。

有些嵌套查询可以用连接运算替代,有些是不能替代的。对于可以用连接运算代替嵌套查询的,到底采用哪种方法用户可以根据自己的习惯确定。

[**例 39**]和[**例 40**]中子查询的查询条件不依赖于父查询,这类子查询称为**不相关子查询**。不相关子查询是较简单的一类子查询。如果子查询的查询条件依赖于父查询,这类子查询称为**相关子查询**(Correlated Subquery),整个查询语句称为**相关嵌套查询**(Correlated nested query)语句。

下面[**例 41**]就是一个相关子查询的例子。

二、带有比较运算符的子查询

带有比较运算符的子查询是指父查询与子查询之间用比较运算符进行连接。当用户能确切知道内层查询返回的是单值时,可以用 >,<,=,>=,<=,!= 或 <> 等比较运算符。

例如在[**例 39**]中,由于一个学生只可能在一个系学习,也就是说内查询的结果是一个值,因此可以用 = 代替 IN:

```
SELECT Sno,Sname,Sdept                    /* [例 39]的解法二 */
FROM Student
WHERE Sdept =
      (SELECT Sdept
       FROM Student
       WHERE Sname =' 刘晨 ');
```

需要注意的是,子查询一定要跟在比较符之后,下列写法是错误的:

```
SELECT Sno,Sname,Sdept
FROM Student
WHERE (SELECT Sdept
       FROM Student
```

```
WHERE Sname = '刘晨' ) = Sdept;
```

[例 41] 找出每个学生超过他选修课程平均成绩的课程号。

```
SELECT Sno, Cno
FROM SC x
WHERE Grade >= (SELECT AVG(Grade)                    /* 某学生的平均成绩 */
                FROM SC y
                WHERE y.Sno = x.Sno);
```

x 是表 SC 的别名，又称为元组变量，可以用来表示 SC 的一个元组。内层查询是求一个学生所有选修课程平均成绩的，至于是哪个学生的平均成绩要看参数 x.Sno 的值，而该值是与父查询相关的，因此这类查询称为**相关子查询**。

这个语句的一种可能的执行过程是：

1. 从外层查询中取出 SC 的一个元组 x，将元组 x 的 Sno 值(200215121)传送给内层查询。

```
SELECT AVG(Grade)
FROM SC y
WHERE y.Sno = '200215121';
```

2. 执行内层查询，得到值 88(近似值)，用该值代替内层查询，得到外层查询：

```
SELECT Sno, Cno
FROM SC x
WHERE Grade >= 88;
```

3. 执行这个查询，得到

(200215121,1)

(200215121,3)

然后外层查询取出下一个元组重复做上述 1 至 3 步骤的处理，直到外层的 SC 元组全部处理完毕。结果为

(200215121,1)
(200215121,3)
(200215122,2)

求解相关子查询不能像求解不相关子查询那样，一次将子查询求解出来，然后求解父查询。内层查询由于与外层查询有关，因此必须反复求值。

三、带有 ANY(SOME)或 ALL 谓词的子查询

子查询返回单值时可以用比较运算符，但返回多值时要用 ANY(有的系统用 SOME)或 ALL 谓词修饰符。而使用 ANY 或 ALL 谓词时则必须同时使用比较运算符。其语义为：

> ANY	大于子查询结果中的某个值
> ALL	大于子查询结果中的所有值
< ANY	小于子查询结果中的某个值
< ALL	小于子查询结果中的所有值
>= ANY	大于等于子查询结果中的某个值
>= ALL	大于等于子查询结果中的所有值
<= ANY	小于等于子查询结果中的某个值
<= ALL	小于等于子查询结果中的所有值
= ANY	等于子查询结果中的某个值
= ALL	等于子查询结果中的所有值(通常没有实际意义)
!= (或 <>) ANY	不等于子查询结果中的某个值
!= (或 <>) ALL	不等于子查询结果中的任何一个值

[例 42] 查询其他系中比计算机科学系某一学生年龄小的学生姓名和年龄。

```
SELECT Sname,Sage
FROM Student
WHERE Sage < ANY ( SELECT Sage
                   FROM Student
                   WHERE Sdept = ' CS ')
  AND Sdept  <> ' CS ';                    /* 注意这是父查询块中的条件 */;
```

结果如下:

Sname	Sage
王敏	18
张立	19

RDBMS 执行此查询时,首先处理子查询,找出 CS 系中所有学生的年龄,构成一个集合(20,19)。然后处理父查询,找所有不是 CS 系且年龄小于 20 或 19 的学生。

本查询也可以用聚集函数来实现。首先用子查询找出 CS 系中最大年龄(20),然后在父查询中查所有非 CS 系且年龄小于 20 岁的学生。SQL 语句如下:

```
SELECT Sname,Sage
FROM Student
WHERE Sage <
            (SELECT MAX(Sage)
             FROM Student
             WHERE Sdept = ' CS ')
  AND Sdept <> ' CS ';
```

[例 43] 查询其他系中比计算机科学系所有学生年龄都小的学生姓名及年龄。

```
SELECT Sname,Sage
FROM Student
```

```
WHERE Sage < ALL
            (SELECT Sage
             FROM Student
             WHERE Sdept = 'CS')
  AND Sdept <> 'CS';
```

RDBMS 执行此查询时,首先处理子查询,找出 CS 系中所有学生的年龄,构成一个集合(20,19)。然后处理父查询,找所有不是 CS 系且年龄小于 20,也小于 19 的学生。查询结果为:

Sname	Sage
王敏	18

本查询同样也可以用聚集函数实现。SQL 语句如下:

```
SELECT Sname,Sage
FROM Student
WHERE Sage <
            (SELECT MIN(Sage)
             FROM Student
             WHERE Sdept = 'CS')
  AND Sdept <>'CS';
```

事实上,用聚集函数实现子查询通常比直接用 ANY 或 ALL 查询效率要高。ANY、ALL 与聚集函数的对应关系如表 3.5 所示。

表 3.5 ANY(或 SOME),ALL 谓词与聚集函数、IN 谓词的等价转换关系

	=	<> 或 !=	<	<=	>	>=
ANY	IN	--	< MAX	<= MAX	> MIN	>= MIN
ALL	--	NOT IN	< MIN	<= MIN	> MAX	>= MAX

表 3.5 中,= ANY 等价于 IN 谓词,< ANY 等价于 < MAX,<> ALL 等价于 NOT IN 谓词,< ALL 等价于 < MIN,等。

四、带有 EXISTS 谓词的子查询

EXISTS 代表存在量词 ∃。带有 EXISTS 谓词的子查询不返回任何数据,只产生逻辑真值"true"或逻辑假值"false"。

可以利用 EXISTS 来判断 $x \in S$、$S \subseteq R$、$S = R$、$S \cap R$ 非空等是否成立。

[例 44] 查询所有选修了 1 号课程的学生姓名。

本查询涉及 Student 和 SC 表。可以在 Student 中依次取每个元组的 Sno 值,用此值去检查 SC 表。若 SC 中存在这样的元组,其 Sno 值等于此 Student.Sno 值,并且其 Cno = '1',则取此 Student.Sname 送入结果表。将此想法写成 SQL 语句是:

```
SELECT Sname
FROM Student
WHERE EXISTS
        (SELECT *
         FROM SC
         WHERE Sno = Student. Sno AND Cno = '1');
```

使用存在量词 EXISTS 后，若内层查询结果非空，则外层的 WHERE 子句返回真值，否则返回假值。

由 EXISTS 引出的子查询，其目标列表达式通常都用 *，因为带 EXISTS 的子查询只返回真值或假值，给出列名无实际意义。

本例中子查询的查询条件依赖于外层父查询的某个属性值（在本例中是 Student 的 Sno 值），因此也是相关子查询。这个相关子查询的处理过程是：首先取外层查询中（Student）表的第一个元组，根据它与内层查询相关的属性值（Sno 值）处理内层查询，若 WHERE 子句返回值为真，则取外层查询中该元组的 Sname 放入结果表；然后再取（Student）表的下一个元组；重复这一过程，直至外层（Student）表全部检查完为止。

本例中的查询也可以用连接运算来实现，读者可以参照有关的例子，自己给出相应的 SQL 语句。

与 EXISTS 谓词相对应的是 NOT EXISTS 谓词。使用存在量词 NOT EXISTS 后，若内层查询结果为空，则外层的 WHERE 子句返回真值，否则返回假值。

[例 45]　查询没有选修 1 号课程的学生姓名。

```
SELECT Sname
FROM Student
WHERE NOT EXISTS
        (SELECT *
         FROM SC
         WHERE Sno = Student. Sno AND Cno = '1') ;
```

一些带 EXISTS 或 NOT EXISTS 谓词的子查询不能被其他形式的子查询等价替换，但所有带 IN 谓词、比较运算符、ANY 和 ALL 谓词的子查询都能用带 EXISTS 谓词的子查询等价替换。例如带有 IN 谓词的[例 39]可以用如下带 EXISTS 谓词的子查询替换：

```
SELECT Sno, Sname, Sdept                    /* [例 39]的解法四 */
FROM Student S1
WHERE EXISTS
        (SELECT *
          FROM Student S2
          WHERE S2. Sdept = S1. Sdept AND
                S2. Sname = '刘晨');
```

由于带 EXISTS 量词的相关子查询只关心内层查询是否有返回值，并不需要查具体值，因此其效率并不一定低于不相关子查询，有时是高效的方法。

[例 46] 查询选修了全部课程的学生姓名。

SQL 中没有全称量词∀(For all)。但是，我们总可以把带有全称量词的谓词转换为等价的带有存在量词的谓词：

$$(\forall x)P \equiv \neg(\exists x(\neg P))$$

由于没有全称量词，可将题目的意思转换成等价的用存在量词的形式：查询这样的学生，没有一门课程是他不选修的。其 SQL 语句为：

```
SELECT Sname
FROM Student
WHERE NOT EXISTS
        (SELECT *
         FROM Course
         WHERE NOT EXISTS
                (SELECT *
                 FROM SC
                 WHERE Sno = Student. Sno
                    AND Cno = Course. Cno));
```

从而用 EXIST/NOT EXIST 来实现带全称量词的查询。

[例 47] 查询至少选修了学生 200215122 选修的全部课程的学生号码。

本查询可以用逻辑蕴涵来表达：查询学号为 x 的学生，对所有的课程 y，只要 200215122 学生选修了课程 y，则 x 也选修了 y。形式化表示如下：

用 p 表示谓词“学生 200215122 选修了课程 y”

用 q 表示谓词“学生 x 选修了课程 y”

则上述查询为：$(\forall y)\ p \rightarrow q$

SQL 语言中没有蕴涵(Implication)逻辑运算。但是，可以利用谓词演算将一个逻辑蕴涵的谓词等价转换为：$p \rightarrow q \equiv \neg p \vee q$

该查询可以转换为如下等价形式：

$$(\forall y)p \rightarrow q \equiv \neg(\exists y(\neg(p \rightarrow q)) \equiv \neg(\exists y(\neg(\neg p \vee q) \equiv \neg \exists y(p \wedge \neg q)$$

它所表达的语义为：不存在这样的课程 y，学生 200215122 选修了 y，而学生 x 没有选。用 SQL 语言表示如下：

```
SELECT DISTINCT Sno
FROM SC SCX
WHERE NOT EXISTS
        (SELECT *
         FROM SC SCY
```

```
WHERE SCY. Sno = '200215122' AND
      NOT EXISTS
      (SELECT *
       FROM SC SCZ
       WHERE SCZ. Sno = SCX. Sno AND
             SCZ. Cno = SCY. Cno));
```

3.4.4 集合查询

SELECT 语句的查询结果是元组的集合，所以多个 SELECT 语句的结果可进行集合操作。集合操作主要包括并操作 UNION、交操作 INTERSECT 和差操作 EXCEPT。注意，参加集合操作的各查询结果的列数必须相同；对应项的数据类型也必须相同。

[例 48] 查询计算机科学系的学生及年龄不大于 19 岁的学生。

```
SELECT *
FROM Student
WHERE Sdept = 'CS'
UNION
SELECT *
FROM Student
WHERE Sage <= 19;
```

本查询实际上是求计算机科学系的所有学生与年龄不大于 19 岁的学生的并集。使用 UNION 将多个查询结果合并起来时，系统会自动去掉重复元组。如果要保留重复元组则用 UNION ALL 操作符。

[例 49] 查询选修了课程 1 或者选修了课程 2 的学生。

即查选修课程 1 的学生集合与选修课程 2 的学生集合的并集。

```
SELECT Sno
FROM SC
WHERE Cno = '1'
UNION
SELECT Sno
FROM SC
WHERE Cno = '2';
```

[例 50] 查询计算机科学系的学生与年龄不大于 19 岁的学生的交集。

```
SELECT *
FROM Student
WHERE Sdept = 'CS'
INTERSECT
SELECT *
```

```
FROM Student
WHERE Sage <= 19;
```

这实际上就是查询计算机科学系中年龄不大于 19 岁的学生。

```
SELECT *
FROM Student
WHERE Sdept = 'CS' AND
      Sage <= 19;
```

[例 51] 查询既选修了课程 1 又选修了课程 2 的学生。就是查询选修课程 1 的学生集合与选修课程 2 的学生集合的交集。

```
SELECT Sno
FROM SC
WHERE Cno = '1'
INTERSECT
SELECT Sno
FROM SC
WHERE Cno = '2';
```

本例也可以表示为：

```
SELECT Sno
FROM SC
WHERE Cno = '1' AND Sno IN
                    (SELECT Sno
                     FROM SC
                     WHERE Cno = '2');
```

[例 52] 查询计算机科学系的学生与年龄不大于 19 岁的学生的差集。

```
SELECT *
FROM Student
WHERE Sdept = 'CS'
EXCEPT
SELECT *
FROM Student
WHERE Sage <= 19;
```

也就是查询计算机科学系中年龄大于 19 岁的学生。

```
SELECT *
FROM Student
WHERE Sdept = 'CS' AND Sage > 19;
```

3.4.5 SELECT 语句的一般格式

SELECT 语句是 SQL 的核心语句，从上面的例子可以看到其语句成分丰富多样，下面总结一下它们的一般格式。

SELECT 语句的一般格式：

```
SELECT [ALL|DISTINCT] <目标列表达式> [别名][,<目标列表达式> [别名]]…
FROM <表名或视图名> [别名][,<表名或视图名> [别名]]…
[WHERE <条件表达式>]
[GROUP BY <列名 1> [HAVING <条件表达式>]]
[ORDER BY <列名 2> [ASC|DESC]];
```

1. 目标列表达式有以下可选格式：

(1) *

(2) <表名>.*

(3) COUNT([DISTINCT|ALL] *)

(4) [<表名>.]<属性列名表达式>[,[<表名>.]<属性列名表达式>]…

其中<属性列名表达式>可以是由属性列、作用于属性列的聚集函数和常量的任意算术运算(+,-,*,/)组成的运算公式。

2. 聚集函数的一般格式为：

$$\begin{Bmatrix}\text{COUNT}\\ \text{SUM}\\ \text{AVG}\\ \text{MAX}\\ \text{MIN}\end{Bmatrix}([\text{DISTINCT}|\text{ALL}]\ <\text{列名}>)$$

3. WHERE 子句的条件表达式有以下可选格式：

(1)

$$<\text{属性列名}>\ \theta\begin{Bmatrix}<\text{属性列名}>\\ <\text{常量}>\\ [\text{ANY}|\text{ALL}](\text{SELECT 语句})\end{Bmatrix}$$

(2)

$$<\text{属性列名}>\ [\text{NOT}]\text{BETWEEN}\begin{Bmatrix}<\text{属性列名}>\\ <\text{常量}>\\ (\text{SELECT 语句})\end{Bmatrix}\text{AND}\begin{Bmatrix}<\text{属性列名}>\\ <\text{常量}>\\ (\text{SELECT 语句})\end{Bmatrix}$$

(3)

$$<\text{属性列名}>\ [\text{NOT}]\text{IN}\begin{Bmatrix}(<\text{值 1}>[,<\text{值 2}>]\cdots)\\ (\text{SELECT 语句})\end{Bmatrix}$$

(4) <属性列名> [NOT] LIKE <匹配串>

(5) <属性列名> IS [NOT] NULL

(6) [NOT] EXISTS (SELECT 语句)

(7) <条件表达式> {AND / OR} <条件表达式> [{AND / OR} <条件表达式>] …

3.5 数据更新

数据更新操作有3种:向表中添加若干行数据、修改表中的数据和删除表中的若干行数据。在SQL中有相应的三类语句。

3.5.1 插入数据

SQL的数据插入语句INSERT通常有两种形式。一种是插入一个元组,另一种是插入子查询结果。后者可以一次插入多个元组。

一、插入元组

插入元组的INSERT语句的格式为

```
INSERT
INTO <表名> [(<属性列1>[,<属性列2>…)]
VALUES (<常量1> [,<常量2>]…);
```

其功能是将新元组插入指定表中。其中新元组的属性列1的值为常量1,属性列2的值为常量2,…。INTO子句中没有出现的属性列,新元组在这些列上将取空值。但必须注意的是,在表定义时说明了NOT NULL的属性列不能取空值,否则会出错。

如果INTO子句中没有指明任何属性列名,则新插入的元组必须在每个属性列上均有值。

[例1] 将一个新学生元组(学号:200215128;姓名:陈冬;性别:男;所在系:IS;年龄:18岁)插入到Student表中。

```
INSERT
INTO Student (Sno,Sname,Ssex,Sdept,Sage)
VALUES ('200215128','陈冬','男','IS',18);
```

在INTO子句中指出了表名Student,指出了新增加的元组在哪些属性上要赋值,属性的顺序可以与CREATE TABLE中的顺序不一样。VALUES子句对新元组的各属性赋值,字符串常数要用单引号(英文符号)括起来。

[例2] 将学生张成民的信息插入到Student表中。

```
INSERT
INTO Student
VALUES ('200215126', '张成民', '男',18, 'CS');
```

同[例 1]的不同是在 INTO 子句中只指出了表名,没有指出属性名,这表示新元组要在表的所有属性列上都指定值,属性列的次序与 CREATE TABLE 中的次序相同。VALUES 子句对新元组的各属性列赋值,一定要注意值与属性列要一一对应,如果像[例 1]那样,成为('200215126', '张成民', '男', 'CS',18),则含义是将'CS'赋予了列 Sage,而 18 赋予了列 Sdept,这样则会因为数据类型不同而出错。

[例 3] 插入一条选课记录('200215128','1')。

```
INSERT
INTO SC(Sno,Cno)
VALUES ('200215128','1');
```

RDBMS 将在新插入记录的 Grade 列上自动地赋空值。

或者:

```
INSERT
INTO SC
VALUES ('200215128','1',NULL);
```

因为没有指出 SC 的属性名,在 Grade 列上要明确给出空值。

二、插入子查询结果

子查询不仅可以嵌套在 SELECT 语句中,用以构造父查询的条件(如 3.4.3 节所述),也可以嵌套在 INSERT 语句中,用以生成要插入的批量数据。

插入子查询结果的 INSERT 语句的格式为

```
INSERT
INTO <表名> [(<属性列 1> [,<属性列 2>…)]
子查询;
```

[例 4] 对每一个系,求学生的平均年龄,并把结果存入数据库。

首先在数据库中建立一个新表,其中一列存放系名,另一列存放相应的学生平均年龄。

```
CREATE TABLE Dept_age
        (Sdept CHAR(15)
         Avg_age SMALLINT);
```

然后对 Student 表按系分组求平均年龄,再把系名和平均年龄存入新表中。

```
INSERT
INTO Dept_age(Sdept,Avg_age)
```

```
SELECT Sdept,AVG(Sage)
FROM Student
GROUP BY Sdept;
```

3.5.2 修改数据

修改操作又称为更新操作,其语句的一般格式为

```
UPDATE <表名>
SET <列名>=<表达式>[,<列名>=<表达式>]…
[WHERE <条件>];
```

其功能是修改指定表中满足 WHERE 子句条件的元组。其中 SET 子句给出 <表达式> 的值用子取代相应的属性列值。如果省略 WHERE 子句,则表示要修改表中的所有元组。

一、修改某一个元组的值

[例 5] 将学生 200215121 的年龄改为 22 岁。

```
UPDATE Student
SET Sage = 22
WHERE Sno = '200215121';
```

二、修改多个元组的值

[例 6] 将所有学生的年龄增加 1 岁。

```
UPDATE Student
SET Sage = Sage + 1;
```

三、带子查询的修改语句

子查询也可以嵌套在 UPDATE 语句中,用以构造修改的条件。

[例 7] 将计算机科学系全体学生的成绩置零。

```
UPDATE SC
SET Grade = 0
WHERE 'CS' =
        (SELETE Sdept
         FROM Student
         WHERE Student. Sno = SC. Sno);
```

3.5.3 删除数据

删除语句的一般格式为

```
DELETE
FROM <表名>
[WHERE <条件>];
```

DELETE 语句的功能是从指定表中删除满足 WHERE 子句条件的所有元组。如果省略 WHERE 子句，表示删除表中全部元组，但表的定义仍在字典中。也就是说，DELETE 语句删除的是表中的数据，而不是关于表的定义。

一、删除某一个元组的值

[例 8]　删除学号为 200215128 的学生记录。

```
DELETE
FROM Student
WHERE Sno = '200215128';
```

二、删除多个元组的值

[例 9]　删除所有的学生选课记录。

```
DELETE
FROM SC;
```

这条 DELETE 语句将使 SC 成为空表，它删除了 SC 的所有元组。

三、带子查询的删除语句

子查询同样也可以嵌套在 DELETE 语句中，用以构造执行删除操作的条件。

[例 10]　删除计算机科学系所有学生的选课记录。

```
DELETE
FROM SC
WHERE 'CS' =
        (SELETE Sdept
         FROM Student
         WHERE Student. Sno = SC. Sno);
```

对某个基本表中数据的增、删、改操作有可能会破坏参照完整性，第五章 5.2 将详细讲解如何进行参照完整性检查和控制。

3.6　视　　图

视图是从一个或几个基本表(或视图)导出的表。它与基本表不同，是一个虚表。数据库中只存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中。所以基本表中的数据发生变化，从视图中查询出的数据也就随之改变了。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中自己感兴趣的数据及其变化。

视图一经定义，就可以和基本表一样被查询、被删除。也可以在一个视图之上再定义新的视图，但对视图的更新(增、删、改)操作则有一定的限制。

本节专门讨论视图的定义、操作及优点。

3.6.1 定义视图

一、建立视图

SQL 语言用 CREATE VIEW 命令建立视图,其一般格式为

```
CREATE VIEW <视图名>[(<列名>[,<列名>]…)]
AS <子查询>
[WITH CHECK OPTION];
```

其中,子查询可以是任意复杂的 SELECT 语句,但通常不允许含有 ORDER BY 子句和 DISTINCT 短语。

WITH CHECK OPTION 表示对视图进行 UPDATE,INSERT 和 DELETE 操作时要保证更新、插入或删除的行满足视图定义中的谓词条件(即子查询中的条件表达式)。

组成视图的属性列名或者全部省略或者全部指定,没有第三种选择。如果省略了视图的各个属性列名,则隐含该视图由子查询中 SELECT 子句目标列中的诸字段组成。但在下列三种情况下必须明确指定组成视图的所有列名:

(1) 某个目标列不是单纯的属性名,而是聚集函数或列表达式;

(2) 多表连接时选出了几个同名列作为视图的字段;

(3) 需要在视图中为某个列启用新的更合适的名字。

[例 1] 建立信息系学生的视图。

```
CREATE VIEW IS_Student
AS
SELECT Sno,Sname,Sage
FROM Student
WHERE Sdept = 'IS';
```

本例中省略了视图 IS_Student 的列名,隐含了由子查询中 SELECT 子句中的三个列名组成。

RDBMS 执行 CREATE VIEW 语句的结果只是把视图的定义存入数据字典,并不执行其中的 SELECT 语句。只是在对视图查询时,才按视图的定义从基本表中将数据查出。

[例 2] 建立信息系学生的视图,并要求进行修改和插入操作时仍需保证该视图只有信息系的学生。

```
CREATE VIEW IS_Student
AS
SELECT Sno,Sname,Sage
FROM Student
WHERE Sdept = 'IS'
WITH CHECK OPTION;
```

由于在定义 IS_Student 视图时加上了 WITH CHECK OPTION 子句，以后对该视图进行插入、修改和删除操作时，RDBMS 会自动加上 Sdept = 'IS' 的条件。

若一个视图是从单个基本表导出的，并且只是去掉了基本表的某些行和某些列，但保留了主码，我们称这类视图为**行列子集视图**。IS_Student 视图就是一个行列子集视图。

视图不仅可以建立在单个基本表上，也可以建立在多个基本表上。

[例 3]　建立信息系选修了 1 号课程的学生的视图。

```
CREATE VIEW IS_S1(Sno,Sname,Grade)
AS
SELECT Student.Sno,Sname,Grade
FROM Student,SC
WHERE Sdept = 'IS' AND
      Student.Sno = SC.Sno AND
      SC.Cno = '1';
```

由于视图 IS_S1 的属性列中包含了 Student 表与 SC 表的同名列 Sno，所以必须在视图名后面明确说明视图的各个属性列名。

视图不仅可以建立在一个或多个基本表上，也可以建立在一个或多个已定义好的视图上，或建立在基本表与视图上。

[例 4]　建立信息系选修了 1 号课程且成绩在 90 分以上的学生的视图。

```
CREATE VIEW IS_S2
AS
SELECT Sno,Sname,Grade
FROM IS_S1
WHERE Grade >= 90;
```

这里的视图 IS_S2 就是建立在视图 IS_S1 之上的。

定义基本表时，为了减少数据库中的冗余数据，表中只存放基本数据，由基本数据经过各种计算派生出的数据一般是不存储的。但由于视图中的数据并不实际存储，所以定义视图时可以根据应用的需要，设置一些派生属性列。这些派生属性由于在基本表中并不实际存在也称它们为**虚拟列**。带虚拟列的视图也称为**带表达式的视图**。

[例 5]　定义一个反映学生出生年份的视图。

```
CREATE VIEW BT_S(Sno,Sname,Sbirth)
AS
SELECT Sno,Sname,2004 - Sage
FROM Student;
```

BT_S 视图是一个带表达式的视图。视图中的出生年份值是通过计算得到的。

还可以用带有聚集函数和 GROUP BY 子句的查询来定义视图，这种视图称为**分组视图**。

[例 6] 将学生的学号及他的平均成绩定义为一个视图。

```
CREAT VIEW S_G(Sno,Gavg)
AS
SELECT Sno,AVG(Grade)
FROM SC
GROUP BY Sno;
```

由于 AS 子句中 SELECT 语句的目标列平均成绩是通过作用聚集函数得到的,所以 CREATE VIEW 中必须明确定义组成 S_G 视图的各个属性列名,S_G 是一个分组视图。

[例 7] 将 Student 表中所有女生记录定义为一个视图。

```
CREATE VIEW F_Student(F_sno,name,sex,age,dept)
AS
SELECT *
FROM Student
WHERE Ssex ='女';
```

这里视图 F_Student 是由子查询"SELECT *"建立的。F_Student 视图的属性列与 Student 表的属性列一一对应。如果以后修改了基本表 Student 的结构,则 Student 表与 F_Student视图的映像关系就被破坏了,该视图就不能正确工作了。为避免出现这类问题,最好在修改基本表之后删除由该基本表导出的视图,然后重建这个视图。

二、删除视图

该语句的格式为:

```
DROP VIEW <视图名> [CASCADE];
```

视图删除后视图的定义将从数据字典中删除。如果该视图上还导出了其他视图,则使用 CASCADE 级联删除语句,把该视图和由它导出的所有视图一起删除。

基本表删除后,由该基本表导出的所有视图(定义)没有被删除,但均已无法使用了。删除这些视图(定义)需要显式地使用 DROP VIEW 语句。

[例 8] 删除视图 BT_S:DROP VIEW BT_S;

删除视图 IS_S1:DROP VIEW IS_S1;

执行此语句时由于 IS_S1 视图上还导出了 IS_S2 视图,所以该语句被拒绝执行。如果确要删除,则使用级联删除语句:

```
DROP VIEW IS_S1 CASCADE;        /* 删除了视图 IS_S1 和由它导出的所有视图 */
```

3.6.2 查询视图

视图定义后,用户就可以像对基本表一样对视图进行查询了。

[例 9] 在信息系学生的视图中找出年龄小于 20 岁的学生。

```
SELECT Sno,Sage
FROM IS_Student
WHERE Sage < 20;
```

RDBMS 执行对视图的查询时,首先进行有效性检查。检查查询中涉及的表、视图等是否存在。如果存在,则从数据字典中取出视图的定义,把定义中的子查询和用户的查询结合起来,转换成等价的对基本表的查询,然后再执行修正了的查询。这一转换过程称为**视图消解**(View Resolution)。

本例转换后的查询语句为:

```
SELECT Sno,Sage
FROM Student
WHERE Sdept = 'IS' AND Sage < 20;
```

[**例 10**] 查询选修了 1 号课程的信息系学生。

```
SELECT IS_Student. Sno,Sname
FROM IS_Student,SC
WHERE IS_Student. Sno = SC. Sno AND SC. Cno = '1';
```

本查询涉及视图 IS_Student(虚表)和基本表 SC,通过这两个表的连接来完成用户请求。

在一般情况下,视图查询的转换是直接了当的。但有些情况下,这种转换不能直接进行,查询时就会出现问题,如[**例 11**]。

[**例 11**] 在 S_G 视图([**例 6**]中定义的视图)中查询平均成绩在 90 分以上的学生学号和平均成绩,语句为

```
SELECT *
FROM S_G
WHERE Gavg >= 90;
```

[**例 6**]中定义 S_G 视图的子查询为

```
SELECT Sno,AVG(Grade)
FROM SC
GROUP BY Sno;
```

将本例中的查询语句与定义 S_G 视图的子查询结合,形成下列查询语句:

```
SELECT Sno,AVG(Grade)
FROM SC
WHERE AVG(Grade) >= 90
GROUP BY Sno;
```

因为 WHERE 子句中是不能用聚集函数作为条件表达式的,因此执行此修正后的查询

将会出现语法错误。正确转换的查询语句应该是：

```
SELECT Sno,AVG(Grade)
FROM SC
GROUP BY Sno
HAVING AVG(Grade) >=90;
```

目前多数关系数据库系统对行列子集视图的查询均能进行正确转换。但对非行列子集视图的查询(如[例 11])就不一定能做转换了,因此这类查询应该直接对基本表进行。

3.6.3 更新视图

更新视图是指通过视图来插入(INSERT)、删除(DELETE)和修改(UPDATE)数据。

由于视图是不实际存储数据的虚表,因此对视图的更新,最终要转换为对基本表的更新。像查询视图那样,对视图的更新操作也是通过视图消解,转换为对基本表的更新操作。

为防止用户通过视图对数据进行增加、删除、修改时,有意无意地对不属于视图范围内的基本表数据进行操作,可在定义视图时加上 WITH CHECK OPTION 子句。这样在视图上增删改数据时,RDBMS 会检查视图定义中的条件,若不满足条件,则拒绝执行该操作。

[例 12] 将信息系学生视图 IS_Student 中学号为 200215122 的学生姓名改为"刘辰"。

```
UPDATE IS_Student
SET Sname ='刘辰'
WHERE Sno ='200215122';
```

转换后的更新语句为：

```
UPDATE Student
SET Sname ='刘辰'
WHERE Sno ='200215122' AND Sdept ='IS';
```

[例 13] 向信息系学生视图 IS_Student 中插入一个新的学生记录,其中学号为 200215129,姓名为赵新,年龄为 20 岁。

```
INSERT
INTO IS_Student
VALUES('200215129','赵新',20);
```

转换为对基本表的更新：

```
INSERT
INTO Student(Sno,Sname,Sage,Sdept)
VALUES('200215129','赵新',20,'IS');
```

这里系统自动将系名'IS'放入 VALUES 子句中。

[例 14] 删除信息系学生视图 IS_Student 中学号为 200215129 的记录

```
DELETE
FROM IS_Student
WHERE Sno = '200215129';
```

转换为对基本表的更新：

```
DELETE
FROM Student
WHERE Sno = '200215129' AND Sdept = 'IS';
```

在关系数据库中，并不是所有的视图都是可更新的，因为有些视图的更新不能唯一地有意义地转换成对相应基本表的更新。

例如，前面定义的视图 S_G[例 6]是由学号和平均成绩两个属性列组成的，其中平均成绩一项是由 Student 表中对元组分组后计算平均值得来的：

```
CREAT VIEW S_G(Sno,Gavg)
AS
SELECT Sno,AVG(Grade)
FROM SC
GROUP BY Sno;
```

如果想把视图 S_G 中学号为 200215121 的学生的平均成绩改成 90 分，SQL 语句如下：

```
UPDATE S_G
SET Gavg = 90
WHERE Sno = '200215121';
```

但这个对视图的更新是无法转换成对基本表 SC 的更新的，因为系统无法修改各科成绩，以使平均成绩成为 90。所以 S_G 视图是不可更新的。

一般地，行列子集视图是可更新的。除行列子集视图外，还有些视图理论上是可更新的，但它们的确切特征还是尚待研究的课题。还有些视图从理论上就是不可更新的。

目前各个关系数据库系统一般都只允许对行列子集视图进行更新，而且各个系统对视图的更新还有更进一步的规定，由于各系统实现方法上的差异，这些规定也不尽相同。

例如 DB2 规定：

(1) 若视图是由两个以上基本表导出的，则此视图不允许更新。

(2) 若视图的字段来自字段表达式或常数，则不允许对此视图执行 INSERT 和 UPDATE 操作，但允许执行 DELETE 操作。

(3) 若视图的字段来自聚集函数，则此视图不允许更新。

(4) 若视图定义中含有 GROUP BY 子句，则此视图不允许更新。

(5) 若视图定义中含有 DISTINCT 短语，则此视图不允许更新。

(6) 若视图定义中有嵌套查询,并且内层查询的 FROM 子句中涉及的表也是导出该视图的基本表,则此视图不允许更新。例如将 SC 中成绩在平均成绩之上的元组定义成一个视图 GOOD_SC:

```
CREATE VIEW GOOD_SC
AS
SELECT Sno,Cno,Grade
FROM SC
WHERE Grade >
            (SELECT AVG(Grade)
             FROM SC);
```

导出视图 GOOD_SC 的基本表是 SC,内层查询中涉及的表也是 SC,所以视图 GOOD_SC 是不允许更新的。

(7) 一个不允许更新的视图上定义的视图也不允许更新。

应该指出的是,不可更新的视图与不允许更新的视图是两个不同的概念。前者指理论上已证明其是不可更新的视图。后者指实际系统中不支持其更新,但它本身有可能是可更新的视图。

3.6.4 视图的作用

视图最终是定义在基本表之上的,对视图的一切操作最终也要转换为对基本表的操作。而且对于非行列子集视图进行查询或更新时还有可能出现问题。既然如此,为什么还要定义视图呢?这是因为合理使用视图能够带来许多好处。

1. 视图能够简化用户的操作

视图机制使用户可以将注意力集中在所关心的数据上。如果这些数据不是直接来自基本表,则可以通过定义视图,使数据库看起来结构简单、清晰,并且可以简化用户的数据查询操作。例如,那些定义了若干张表连接的视图,就将表与表之间的连接操作对用户隐蔽起来了。换句话说,用户所做的只是对一个虚表的简单查询,而这个虚表是怎样得来的,用户无需了解。

2. 视图使用户能以多种角度看待同一数据

视图机制能使不同的用户以不同的方式看待同一数据,当许多不同种类的用户共享同一个数据库时,这种灵活性是非常重要的。

3. 视图对重构数据库提供了一定程度的逻辑独立性

第一章中已经介绍过数据的物理独立性与逻辑独立性的概念。数据的物理独立性是指用户的应用程序不依赖于数据库的物理结构。数据的逻辑独立性是指当数据库重构造时,如增加新的关系或对原有关系增加新的字段等,用户的应用程序不会受影响。层次数据库和网状数据库一般能较好地支持数据的物理独立性,而对于逻辑独立性则不能完全地支持。

在关系数据库中,数据库的重构往往是不可避免的。重构数据库最常见的是将一个基本表"垂直"地分成多个基本表。例如:将学生关系

Student(Sno,Sname,Ssex,Sage,Sdept)

分为 SX(Sno,Sname,Sage)和 SY(Sno,Ssex,Sdept)两个关系。这时原表 Student 为 SX 表和 SY 表自然连接的结果。如果建立一个视图 Student:

```
CREATE VIEW Student(Sno,Sname,Ssex,Sage,Sdept)
AS
SELECT SX. Sno,SX. Sname,SY. Ssex,SX. Sage,SY. Sdept
FROM SX,SY
WHERE SX. Sno = SY. Sno;
```

这样尽管数据库的逻辑结构改变了(变为 SX 和 SY 两个表了),但应用程序不必修改,因为新建立的视图定义为用户原来的关系,使用户的外模式保持不变,用户的应用程序通过视图仍然能够查找数据。

当然,视图只能在一定程度上提供数据的逻辑独立性,比如由于对视图的更新是有条件的,因此应用程序中修改数据的语句可能仍会因基本表结构的改变而改变。

4. 视图能够对机密数据提供安全保护

有了视图机制,就可以在设计数据库应用系统时,对不同的用户定义不同的视图,使机密数据不出现在不应看到这些数据的用户视图上。这样视图机制就自动提供了对机密数据的安全保护功能。例如 Student 表涉及全校 15 个院系的学生数据,可以在其上定义 15 个视图,每个视图只包含一个院系的学生数据,并只允许每个院系的主任查询和修改自己系的学生视图。

5. 适当的利用视图可以更清晰的表达查询

例如经常需要执行这样的查询"对每个同学找出他获得最高成绩的课程号"。可以先定义一个视图,求出每个同学获得的最高成绩:

```
CREATE VIEW VMGRADE
AS
SELECT Sno, MAX(Grade) Mgrade
FROM SC
GROUP BY Sno;
```

然后用如下的查询语句完成查询:

```
SELECT SC. Sno, Cno
FROM SC, VMGRADE
WHERE SC. Sno = VMGRADE. Sno AND SC. Grade = VMGRADE . Mgrade;
```

3.7 小 结

SQL 可以分为数据定义、数据查询、数据更新、数据控制四大部分。人们有时把数据更新称为数据操纵，或把数据查询与数据更新合称为数据操纵。本章系统而详尽地讲解了前面三部分的内容。

数据控制中的数据安全性和完整性控制将放在第五章和第六章中讲解。嵌入式 SQL、过程性 SQL 与存储过程放在第七章中讲解。

在讲解 SQL 的同时进一步讲解了关系数据库系统的基本概念，使这些概念更加具体、更加丰富。

SQL 是关系数据库语言的工业标准。目前，大部分数据库管理系统产品都能支持 SQL92，但是许多数据库系统只支持 SQL99 的部分特征，至今尚没有一个数据库系统能够完全支持 SQL99。

SQL 的数据查询功能是最丰富，也是最复杂的，读者应加强实验练习。

习 题

1. 试述 SQL 的特点。
2. 试述 SQL 的定义功能。
3. 用 SQL 语句建立第二章习题 5 中的 4 个表。
4. 针对上题中建立的四个表试用 SQL 完成第二章习题 5 中的查询。
5. 针对习题 3 中的四个表试用 SQL 完成以下各项操作：

(1) 找出所有供应商的姓名和所在城市；

(2) 找出所有零件的名称、颜色、重量；

(3) 找出使用供应商 S1 所供应零件的工程号码；

(4) 找出工程项目 J2 使用的各种零件的名称及其数量；

(5) 找出上海厂商供应的所有零件号码；

(6) 找出使用上海产的零件的工程名称；

(7) 找出没有使用天津产的零件的工程号码；

(8) 把全部红色零件的颜色改成蓝色；

(9) 由 S5 供给 J4 的零件 P6 改为由 S3 供应，请作必要的修改；

(10) 从供应商关系中删除 S2 的记录，并从供应情况关系中删除相应的记录；

(11) 请将 (S2，J6，P4，200) 插入供应情况关系。

6. 什么是基本表？什么是视图？两者的区别和联系是什么？
7. 试述视图的优点。
8. 所有的视图是否都可以更新？为什么？

9. 哪类视图是可以更新的？哪类视图是不可更新的？各举一例说明。

10. 试述某个你熟悉的实际系统中对视图更新的规定。

11. 请为三建工程项目建立一个供应情况的视图，包括供应商代码(SNO)、零件代码(PNO)、供应数量(QTY)。针对该视图完成下列查询：

(1) 找出三建工程项目使用的各种零件代码及其数量；

(2) 找出供应商 S1 的供应情况。

实验 2 交互式 SQL

一、实验目的

熟悉通过 SQL 对数据库进行操作。完成作业的 SQL 练习。

二、实验工具

利用你在实验 1 中安装的 RDBMS 及其交互查询工具来操作 SQL 语言。如果你安装的是 Kingbase ES，其交互查询工具是 ISQLW。

三、实验内容和要求

1. 在 RDBMS 中建立一个数据库，进行实验所要求的各种操作，所有的 SQL 操作均在此建立的新库里进行。

2. 根据以下要求认真进行实验，记录所有的实验用例。

数据定义：基本表的创建、修改及删除；索引的创建和删除；视图的创建和删除。

数据操作：完成各类查询操作(单表查询，连接查询，嵌套查询，集合查询)；完成各类更新操作(插入数据，修改数据，删除数据)。

视图的操作：视图的定义(创建和删除)，查询，更新(注意更新的条件)。

四、实验报告

《数据库系统概论》实验报告				
题目：		姓名	日期	
实验环境：				
实验内容与完成情况：				
出现的问题：				
解决方案(列出遇到的问题和解决办法，列出没有解决的问题)：				

本章参考文献

1 Boyce R, Chamberlin D D, Hammer M, et al. Specifying Queries as Relational Expressions. CACM, 18:11, 1975

(SQL 原名 SEQUEL，是 Boyce 等人 1975 年在文献[1]提出的 SQUARE (Specifying Queries as Relational Expressions)语言的基础上发展起来的。文献[2]对 SQUARE 语言的语法进行了修改，形成了 SEQUEL。文献[3]对 SEQUEL 进行了进一步的改进，最后形成了 SQL。SEQUEL 最早用于 IBM San Jose

研究中心设计的关系数据库系统 SYSTEM R）

2 Chamberlin D D, Boyce R. SEQUEL: A Structured English Query Language. Proceedings of ACM SIGMOD Workshop on Data Description, Access and Control, 1974

3 Chamberlin D D, et al. SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control, IBM Journal of Research and Development. 20, No. 6, 1976

4 Reisner P. Use of Psychological Experimentation as an Aid to Development of a Query Language. TSE, 3: 3, 1977
（文献[4]和[5]分析了 SQL 的优缺点）

5 Date C J. A Critique of the SQL Database Language. ACM SIGMOD Record 14, No. 3, 1984

6 Date C J. A Guide to DB2. Reading, Mass, Addison-Wesley, 1984
（C. J. Date 在文献[6]中描述了 DB2 系统中的 SQL。）

7 [美] Date C J. 刘伶, 张秉训译. DB2 指南. 北京: 科学出版社, 1990

8 X3H2 (American National Standards Database Committee). American National Standard Database Language SQL: Working Draft. Document X3H2-85-1, 1984

9 ANSI: The Database Language SQL, Document ANSI X3. 315, 1986
（1986 年美国国家标准局 ANSI 定义了 SQL 标准，详见文献[8]。）

10 Grifiths P P, Wade B W. An Authorization Mechanism for a Relational Database System. ACM TODS, Vol. 1, No. 3, Sept. 1976

11 Chamberlin D D. A Summary of User Experience with the SQL Data Sublanguage. Proc. International Conference on Databases, Aberdeen, Scotland, 1980

12 Date C J. The Outer Join. in Proceedings of 2nd International Conference on Database (ICOD-2), Cambridge, England, September 1983

13 Dayal U, Bernstein P. On the Updatability of Relational Views. in Proceedings of VLDB, 1978
（[12]和[13]讨论了 SQL 语言的视图修改问题。）

14 Keller A M. Updates to Relational Databases Through Views Involving Joins. In Improving Database Usability and Responsiveness (ed. P. Scheuermann). New York: Academic Press, 1982

15 王珊, 朱青. 数据库系统概论学习指导与习题解答. 北京: 高等教育出版社, 2003

16 Dennis Shasha and Phillipe Bonnet: Database Tuning: Principles Experiments and Troubleshooting Techniques. Morgan Kaufmann Publishers. 2002 (released in June 2002). TEXT. NOTE: the Coop has the books now.

第四章 数据库安全性

在第一章中已经讲到数据库的特点之一是 DBMS 提供统一的数据保护功能来保证数据的安全可靠和正确有效，数据库的数据保护主要包括数据的安全性和数据的完整性。这一章讨论数据的安全性，下一章讨论数据的完整性。

4.1 计算机安全性概述

数据库的安全性是指保护数据库以防止不合法的使用所造成的数据泄露、更改或破坏。

安全性问题不是数据库系统所独有的，所有计算机系统都有这个问题。只是在数据库系统中大量数据集中存放，而且为许多最终用户直接共享，从而使安全性问题更为突出。系统安全保护措施是否有效是数据库系统的主要技术指标之一。

数据库的安全性和计算机系统的安全性，包括计算机硬件、操作系统、网络系统等的安全性，是紧密联系、相互支持的，因此在讨论数据库的安全性之前首先讨论计算机系统安全性的一般问题。

4.1.1 计算机系统的三类安全性问题

所谓**计算机系统安全性**，是指为计算机系统建立和采取的各种安全保护措施，以保护计算机系统中的硬件、软件及数据，防止其因偶然或恶意的原因使系统遭到破坏，数据遭到更改或泄露等。

计算机安全不仅涉及到计算机系统本身的技术问题、管理问题，还涉及法学、犯罪学、心理学的问题。其内容包括了计算机安全理论与策略、计算机安全技术、安全管理、安全评价、安全产品以及计算机犯罪与侦察、计算机安全法律、安全监察等。概括起来，计算机系统的安全性问题可分为三大类，即技术安全类、管理安全类和政策法律类。

技术安全是指计算机系统中采用具有一定安全性的硬件、软件来实现对计算机系统及其所存数据的安全保护，当计算机系统受到无意或恶意的攻击时仍能保证系统正常运行，保证系统内的数据不增加、不丢失、不泄露。

管理安全是指由于管理不善导致的计算机设备和数据介质的物理破坏、丢失等软硬件意外故障以及场地的意外事故等安全问题。

政策法律则指政府部门建立的有关计算机犯罪、数据安全保密的法律道德准则和政策法规、法令。

本书只讨论技术安全。

随着计算机资源共享和网络技术的应用日益广泛和深入，特别是 Internet 技术的发展，计算机安全性问题越来越得到人们的重视。对各种计算机及其相关产品、信息系统的安全性要求越来越高。为此，在计算机安全技术方面逐步发展建立了一套可信(Trusted)计算机系统的概念和标准。只有建立了完善的可信或安全标准，才能规范和指导安全计算机系统部件的生产，比较准确地测定产品的安全性能指标，满足民用和国防的需要。

4.1.2 安全标准简介

计算机以及信息安全技术方面有一系列的安全标准，最有影响的当推 TCSEC 和 CC 这两个标准。

TCSEC 是指 1985 年美国国防部(DoD)正式颁布的《DoD 可信计算机系统评估准则》(Trusted Computer System Evaluation Criteria，TCSEC 或 DoD85)。

在 TCSEC 推出后的十年里，不同的国家都开始启动开发建立在 TCSEC 概念上的评估准则，如欧洲的信息技术安全评估准则(ITSEC)、加拿大的可信计算机产品评估准则(CTCPEC)、美国的信息技术安全联邦标准(FC)草案等。这些准则比 TCSEC 更加灵活，适应了 IT 技术的发展。

为满足全球 IT 市场上互认标准化安全评估结果的需要，CTCPEC、FC、TCSEC 和 ITSEC 的发起组织于 1993 年起开始联合行动，解决原标准中概念和技术上的差异，将各自独立的准则集合成一组单一的、能被广泛使用的 IT 安全准则，这一行动被称为 CC(Common Criteria)项目。项目发起组织的代表建立了专门的委员会来开发 CC，历经多次讨论和修订，CC V2.1 版于 1999 年被 ISO 采用为国际标准，2001 年被我国采用为国家标准。目前 CC 已经基本取代了 TCSEC，成为评估信息产品安全性的主要标准。

上述一系列标准的发展历史如图 4.1 所示。本节简要介绍 TCSEC 和 CC V2.1 的基本内容。

TCSEC 又称桔皮书，1991 年 4 月美国 NCSC(国家计算机安全中心)颁布了《可信计算机系统评估准则关于可信数据库系统的解释》(Trusted Database Interpretation，简称 TDI，即紫皮书)，将 TCSEC 扩展到数据库管理系统。TDI 中定义了数据库管理系统的设计与实现中需满足和用以进行安全性级别评估的标准。

TCSEC/TDI 从以下四个方面来描述安全性级别划分的指标：安全策略、责任、保证和文档。每个方面又细分为若干项。

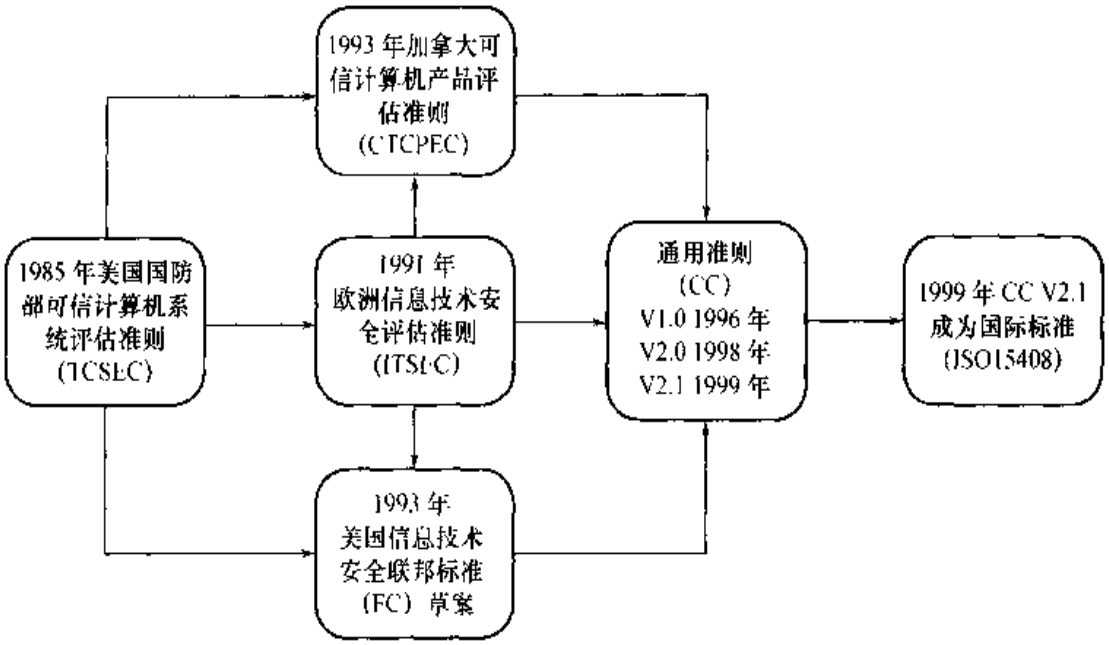


图 4.1 信息安全标准的发展历史

根据计算机系统对各项指标的支持情况，TCSEC(TDI)将系统划分为四组(division)七个等级，依次是 D、C(C1,C2)、B(B1,B2,B3)、A(A1)，按系统可靠或可信程度逐渐增高，如表 4.1 所示。

表 4.1 TCSEC/TDI 安全级别划分

安全级别	定 义
A1	验证设计(Verified Design)
B3	安全域(Security Domains)
B2	结构化保护(Structural Protection)
B1	标记安全保护(Labeled Security Protection)
C2	受控的存取保护(Controlled Access Protection)
C1	自主安全保护(Discretionary Security Protection)
D	最小保护(Minimal Protection)

D 级 D 级是最低级别。保留 D 级的目的是为了将一切不符合更高标准的系统，统统归于 D 组。如 DOS 就是操作系统中安全标准为 D 的典型例子。它具有操作系统的基本功能，如文件系统，进程调度等，但在安全性方面几乎没有什么专门的机制来保障。

C1 级 只提供了非常初级的自主安全保护。能够实现对用户和数据的分离，进行自主存取控制(DAC)，保护或限制用户权限的传播。现有的商业系统往往稍作改进即可满足要求。

C2 级 实际是安全产品的最低档次，提供受控的存取保护，即将 C1 级的 DAC 进一步

细化，以个人身份注册负责，并实施审计和资源隔离。达到C2级的产品在其名称中往往不突出“安全”(Security)这一特色，如操作系统中的Windows 2000，数据库产品中的Oracle 7等。

B1级 标记安全保护。对系统的数据加以标记，并对标记的主体和客体实施强制存取控制(MAC)以及审计等安全机制。B1级别的产品才认为是真正意义上的安全产品，满足此级别的产品前一般多冠以“安全”(Security)或“可信的”(Trusted)字样，作为区别于普通产品的安全产品出售。例如，操作系统方面有惠普公司的HP-UX BLS release 9.0.9+等，数据库方面则有Oracle公司的Trusted Oracle 7，Sybase公司的Secure SQL Server version 11.0.6等。

B2级 结构化保护。建立形式化的安全策略模型并对系统内的所有主体和客体实施DAC和MAC。

B3级 安全域。该级的TCB必须满足访问监控器的要求，审计跟踪能力更强，并提供系统恢复过程。

A1级 验证设计，即提供B3级保护的同时给出系统的形式化设计说明和验证以确信各安全保护真正实现。

CC是在上述各评估准则及具体实践基础上，通过相互总结和互补发展而来的。和早期的评估准则相比，CC具有结构开放、表达方式通用等特点。CC提出了目前国际上公认的表述信息技术安全性的结构，即把对信息产品的安全要求分为安全功能要求和安全保证要求。安全功能要求用以规范产品和系统的安全行为，安全保证要求解决如何正确有效地实施这些功能。安全功能要求和安全保证要求都以“类—子类—组件”的结构表述，组件是安全要求的最小构件块。

CC的文本由三部分组成，三个部分相互依存，缺一不可。

第一部分是“简介和一般模型”，介绍CC中的有关术语、基本概念和一般模型以及与评估有关的一些框架，CC的附录部分主要介绍“保护轮廓”(Protection Profile，简称PP)和“安全目标”(Security Target，简称ST)的基本内容。

第二部分是安全功能要求，列出了一系列功能组件、子类和类。具体来说有11类，分别是安全审计(FAU)、通信(FCO)、密码支持(FCS)、用户数据保护(FDP)、标识和鉴别(FIA)、安全管理(FMT)、隐私(FPR)、TSF保护(FPT)、资源利用(FRU)、TOE访问(FTA)、可信路径和信道(FTP)。这11大类分为66个子类，由135个组件构成。

第三部分是安全保证要求，列出了一系列保证组件、子类和类，包括7个类，分别是配置管理(ACM)、交付和运行(ADO)、开发(ADV)、指导性文档(AGD)、生命周期支持(ALC)、测试(ATE)、脆弱性评定(AVA)。这7大类分为26个子类，由74个组件构成。另外，第三部分中根据系统对安全保证要求的支持情况提出了评估保证级(EAL)，并定义了保护轮廓PP和安全目标ST的评估准则，用于对PP和ST的评估。

这三部分的有机结合具体体现在PP和ST中，CC提出的安全功能要求和安全保证要求都可以在具体的PP和ST中进一步细化和扩展，这种开放式的结构更适应信息安全技术的

发展。CC 的具体应用也是通过 PP 和 ST 这两种结构来实现的。

PP 用于表达一类产品或系统的用户需求，是 CC 在某一领域的具体化。CC 针对不同领域产品的评估，就体现在开发该类产品的 PP 上，例如，针对操作系统产品的 PP(OS PP)、针对数据库产品的 PP(DBMS PP)等。CC 并没有专门针对 DBMS 的解释。DBMS PP 是 CC 在 DBMS 领域的具体化，相当于是对 DBMS 的解释。换句话说，CC 的 DBMS PP 就相当于 TCSEC 的 TDI。CC 中规定了 PP 应包含的基本内容和格式，例如必须指明该 PP 符合哪一种评估保证级别，目前国外已开发多种不同 EAL 的 DBMS PP。ST 是对特定的一种产品进行描述，参加 CC 评估的产品必须提供自己的 ST。

PP 的编制有助于提高安全保护的针对性和有效性。ST 在 PP 的基础上，将安全要求进一步具体化，解决安全要求的具体实现。通过 PP 和 ST 这两种结构，能够方便的将 CC 的安全性要求具体应用到信息产品的开发、生产、测试、评估和信息系统的集成、运行、评估、管理中。

除了 PP 和 ST 之外，还有一个重要的结构是包。包是组件的特定组合，用来描述一组特定的安全功能和保证要求。评估保证级(EAL)是在 CC 第三部分中预先定义的由保证组件组成的保证包，每一保证包描述了一组特定的保证要求，对应着一种评估保证级别。从 EAL1 至 EAL7 共分为七级，按保证程度逐渐增高。如表 4.2 所示。

表 4.2 CC 评估保证级划分

评估保证级	定 义	TCSEC 安全级别(近似相当)
EAL1	功能测试(functionally tested)	
EAL2	结构测试(structurally tested)	C1
EAL3	系统地测试和检查(methodically tested and checked)	C2
EAL4	系统地设计、测试和复查(methodically designed, tested, and reviewed)	B1
EAL5	半形式化设计和测试(semiformally designed and tested)	B2
EAL6	半形式化验证的设计和测试(semiformally verified design and tested)	B3
EAL7	形式化验证的设计和测试(formally verified design and tested)	A1

粗略而言，TCSEC 的 C1 和 C2 级分别相当于 EAL2 和 EAL3；B1、B2 和 B3 分别相当于 EAL4、EAL5 和 EAL6；A1 对应于 EAL7。

目前有许多信息产品，操作系统如 Windows 2000、Sun Solaris 8 等，数据库管理系统如 Oracle9i、DB2 V8.2、Sybase Adaptive Server Enterprise V12.5.2 等，都已通过了 CC 的 EAL4。

有关 CC 的具体要求这里就不详细展开了，有兴趣的读者请参阅本章参考文献[2]或[3]。

4.2 数据库安全性控制

在一般计算机系统中,安全措施是一级一级层层设置的。例如,可以有如下的模型:

在图 4.2 所示的安全模型中,用户要求进入计算机系统时,系统首先根据输入的用户标识进行用户身份鉴定,只有合法的用户才准许进入计算机系统。对已进入系统的用户,DBMS 还要进行存取控制,只允许用户执行合法操作。

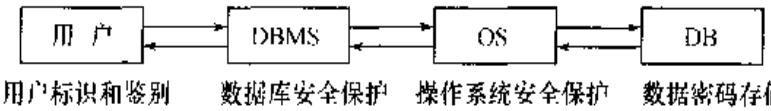


图 4.2 计算机系统的安全模型

操作系统一级也会有自己的保护措施。数据最后还可以以密码形式存储到数据库中。操作系统一级的安全保护措施可参考操作系统的有关书籍,这里不再详叙。另外,对于强力逼迫透露口令、盗窃物理存储设备等行为而采取的保安措施,例如出入机房登记、加锁等,也不在这里讨论之列。

这里只讨论与数据库有关的用户标识和鉴定、存取控制、视图和密码存储等安全技术。

4.2.1 用户标识与鉴别(Identification & Authentication)

用户标识和鉴别是系统提供的最外层安全保护措施。其方法是由系统提供一定的方式让用户标识自己的名字或身份。每次用户要求进入系统时,由系统进行核对,通过鉴定后才提供机器使用权。

对于获得上机权的用户若要使用数据库时数据库管理系统还要进行用户标识和鉴定。用户标识和鉴定的方法有很多种,而且在一个系统中往往是多种方法并举,以获得更强的安全性。常用的方法有:

用户标识(User Identification)

用一个用户名(User Name)或者用户标识号(UID)来标明用户身份。系统内部记录着所有合法用户的标识,系统鉴别此用户是否是合法用户,若是,则可以进入下一步的核实;若不是,则不能使用系统。

口令(Password)

为了进一步核实用户,系统常常要求用户输入口令。为保密起见,用户在终端上输入的口令不显示在屏幕上。系统核对口令以鉴别用户身份。

通过用户名和口令来鉴定用户的方法简单易行,但用户名与口令容易被人窃取,因此还可以用更复杂的方法。例如每个用户都预先约定好一个计算过程或者函数,鉴别用户身份时,系统提供一个随机数,用户根据自己预先约定的计算过程或者函数进行计算,系统根据

用户计算结果是否正确进一步鉴定用户身份。用户可以约定比较简单的计算过程或函数，以便计算起来方便；也可以约定比较复杂的计算过程或函数，以便安全性更好。

用户标识和鉴定可以重复多次。

4.2.2 存取控制

数据库安全性所关心的主要是 DBMS 的存取控制机制。数据库安全最重要的一点就是确保只授权给有资格的用户访问数据库的权限，同时令所有未被授权的人员无法接近数据，这主要通过数据库系统的存取控制机制实现。

存取控制机制主要包括两部分：

1. 定义用户权限，并将用户权限登记到数据字典中

用户对某一数据对象的操作权力称为权限。某个用户应该具有何种权限是个管理问题和政策问题而不是技术问题。DBMS 的功能是保证这些决定的执行。为此 DBMS 系统必须提供适当的语言来定义用户权限，这些定义经过编译后存放在数据字典中，被称做安全规则或授权规则。

2. 合法权限检查

每当用户发出存取数据库的操作请求后（请求一般应包括操作类型、操作对象和操作用户等信息），DBMS 查找数据字典，根据安全规则进行合法权限检查，若用户的操作请求超出了定义的权限，系统将拒绝执行此操作。

用户权限定义和合法权检查机制一起组成了 DBMS 的安全子系统。

前面已经讲到，当前大型的 DBMS 一般都支持 C2 级中的自主存取控制（Discretionary Access Control，简记为 DAC），有些 DBMS 同时还支持 B1 级中的强制存取控制（Mandatory Access Control，简记为 MAC）。

这两类方法的简单定义是：

（1）在**自主存取控制**方法中，用户对于不同的数据库对象有不同的存取权限，不同的用户对同一对象也有不同的权限，而且用户还可将其拥有的存取权限转授给其他用户。因此自主存取控制非常灵活。

（2）在**强制存取控制**方法中，每一个数据库对象被标以一定的密级，每一个用户也被授予某一个级别的许可证。对于任意一个对象，只有具有合法许可证的用户才可以存取。强制存取控制因此相对比较严格。

下面介绍这两种存取控制方法。

4.2.3 自主存取控制（DAC）方法

大型数据库管理系统几乎都支持自主存取控制，目前的 SQL 标准也对自主存取控制提供支持，这主要通过 SQL 的 GRANT 语句和 REVOKE 语句来实现。

用户权限是由两个要素组成的:数据库对象和操作类型。定义一个用户的存取权限就是要定义这个用户可以在哪些数据库对象上进行哪些类型的操作。在数据库系统中,定义存取权限称为**授权**(Authorization)。

在非关系系统中,用户只能对数据进行操作,存取控制的数据库对象也仅限于数据本身。

关系数据库系统中存取控制的对象不仅有数据本身(基本表中的数据、属性列上的数据),还有数据库模式(包括数据库 SCHEMA、基本表 TABLE、视图 VIEW 和索引 INDEX 的创建)等,表 4.3 列出了主要的存取权限。

表 4.3 关系数据库系统中的存取权限

对象类型	对象	操作类型
数据库	模式	CREATE SCHEMA
	基本表	CREATE TABLE,ALTER TABLE
模式	视图	CREATE VIEW
	索引	CREATE INDEX
数据	基本表和视图	SELECT,INSERT,UPDATE,DELETE,REFERENCES, ALL PRIVILEGES
数据	属性列	SELECT,INSERT,UPDATE, REFERENCES ALL PRIVILEGES

4.2.4 授权(Authorization)与回收

某个用户对某类数据库对象具有何种操作权力是个政策问题而不是技术问题。数据库管理系统的功能是保证这些决定的执行。

下面首先讲解 SQL 中的 GRANT 语句和 REVOKE 语句。GRANT 语句向用户授予权限,REVOKE 语句收回授予的权限。

对数据库模式的授权则由 DBA 在创建用户时实现,将在后面讲解。

一、GRANT

GRANT 语句的一般格式为

```
GRANT <权限> [,<权限>]…
ON <对象类型> <对象名>[,<对象类型> <对象名>]…
TO <用户>[,<用户>]…
[WITH GRANT OPTION];
```

其语义为:将对指定操作对象的指定操作权限授予指定的用户。发出该 GRANT 语句的可以是 DBA,也可以是该数据库对象创建者(即属主 Owner),也可以是已经拥有该权限的用

户。接受权限的用户可以是一个或多个具体用户,也可以是 PUBLIC,即全体用户。

如果指定了 WITH GRANT OPTION 子句,则获得某种权限的用户还可以把这种权限再授予其他的用户。如果没有指定 WITH GRANT OPTION 子句,则获得某种权限的用户只能使用该权限,不能传播该权限。

SQL 标准允许具有 WITH GRANT OPTION 的用户把相应权限或其子集传递授予其他用户,但不允许循环授权,即被授权者不能把权限再授回给授权者或其祖先,如图 4.3 所示。

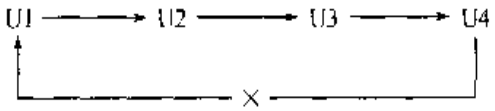


图 4.3 不允许循环授权

[例 1] 把查询 Student 表的权限授给用户 U1。

```
GRANT SELECT
ON TABLE Student
TO U1;
```

[例 2] 把对 Student 表和 Course 表的全部操作权限授予用户 U2 和 U3。

```
GRANT ALL PRIVILEGES
ON TABLE Student,Course
TO U2,U3;
```

[例 3] 把对表 SC 的查询权限授予所有用户。

```
GRANT SELECT
ON TABLE SC
TO PUBLIC;
```

[例 4] 把查询 Student 表和修改学生学号的权限授给用户 U4。

```
GRANT UPDATE(Sno),SELECT
ON TABLE Student
TO U4;
```

这里实际上要授予 U4 用户的是对基本表 Student 的 SELECT 权限和对属性列 Sno 的 UPDATE 权限。对属性列的授权时必须明确指出相应属性列名。

[例 5] 把对表 SC 的 INSERT 权限授予 U5 用户,并允许将此权限再授予其他用户。

```
GRANT INSERT
ON TABLE SC
TO U5
WITH GRANT OPTION;
```

执行此 SQL 语句后，U5 不仅拥有了对表 SC 的 INSERT 权限，还可以传播此权限，即由 U5 用户发上述 GRANT 命令给其他用户。例如 U5 可以将此权限授予 U6：

[例 6] GRANT INSERT
ON TABLE SC
TO U6
WITH GRANT OPTION;

同样，U6 还可以将此权限授予 U7：

[例 7] GRANT INSERT
ON TABLE SC
TO U7;

因为 U6 未给 U7 传播的权限，因此 U7 不能再传播此权限。

由上面的例子可以看到，GRANT 语句可以一次向一个用户授权，如[例 1]所示，这是最简单的一种授权操作；也可以一次向多个用户授权，如[例 2]、[例 3]等所示；还可以一次传播多个同类对象的权限，如[例 2]所示；甚至一次可以完成对基本表和属性列这些不同对象的授权，如[例 4]所示。表 4.4 是执行了[例 1]到[例 7]的语句后学生－课程数据库中的用户权限定义表。

表 4.4 用户权限定义示例

授权用户名	被授权用户名	数据库对象名	允许的操作类型	能否转授权
DBA	U1	关系 Student	SELECT	不能
DBA	U2	关系 Student	ALL	不能
DBA	U2	关系 Course	ALL	不能
DBA	U3	关系 Student	ALL	不能
DBA	U3	关系 Course	ALL	不能
DBA	PUBLIC	关系 SC	SELECT	不能
DBA	U4	关系 Student	SELECT	不能
DBA	U4	属性列 Student. Sno	UPDATE	不能
DBA	U5	关系 SC	INSERT	能
U5	U6	关系 SC	INSERT	能
U6	U7	关系 SC	INSERT	不能

二、REVOKE

授予的权限可以由 DBA 或其他授权者用 REVOKE 语句收回，REVOKE 语句的一般格式为

REVOKE <权限>[,<权限>]…
ON <对象类型> <对象名>[,<对象类型> <对象名>]…

FROM <用户>[,<用户>]…[CASCADE|RESTRICT];

[例 8] 把用户 U4 修改学生学号的权限收回。

```
REVOKE UPDATE(Sno)
ON TABLE Student
FROM U4;
```

[例 9] 收回所有用户对表 SC 的查询权限。

```
REVOKE SELECT
ON TABLE SC
FROM PUBLIC;
```

[例 10] 把用户 U5 对 SC 表的 INSERT 权限收回。

```
REVOKE INSERT
ON TABLE SC
FROM U5 CASCADE;
```

将用户 U5 的 INSERT 权限收回的时候必须级联(CASCADE)收回,不然系统将拒绝(RESTRICT)执行该命令。因为在[例 6]中,U5 将对 SC 表的 INSERT 权限授予了 U6,而 U6 又将其授予了 U7[例 7]。

注:这里缺省值为 RESTRICT,有的 DBMS 缺省值为 CASCADE,会自动执行级联操作而不必明显地写出 CASCADE。如果 U6 或 U7 还从其他用户处获得对 SC 表的 INSERT 权限,则他们仍具有此权限,系统只收回直接或间接从 U5 处获得的权限。

表 4.5 是执行了[例 8]到[例 10]的语句后学生 - 课程数据库中的用户权限定义表。

表 4.5 用户权限定义表示例

授权用户名	被授权用户名	数据库对象名	允许的操作类型	能否转授权
DBA	U1	关系 Student	SELECT	不能
DBA	U2	关系 Student	ALL	不能
DBA	U2	关系 Course	ALL	不能
DBA	U3	关系 Student	ALL	不能
DBA	U3	关系 Course	ALL	不能
DBA	U4	关系 Student	SELECT	不能

SQL 提供了非常灵活的授权机制。DBA 拥有对数据库中所有对象的所有权限,并可以根据实际情况将不同的权限授予不同的用户。

用户对自己建立的基本表和视图拥有全部的操作权限,并且可以用 GRANT 语句把其中某些权限授予其他用户。被授权的用户如果有“继续授权”的许可,还可以把获得的权限再

授予其他用户。

所有授予出去的权力在必要时又都可以用 REVOKE 语句收回。

可见,用户可以"自主"地决定将数据的存取权限授予何人、决定是否也将"授权"的权限授予别人。因此我们称这样的存取控制是**自主存取控制**。

三、创建数据库模式的权限

GRANT 和 REVOKE 语句向用户授予或收回对数据的操作权限。对数据库模式的授权则由 DBA 在创建用户时实现。

CREATE USER 语句一般格式如下:

```
CREATE USER <username>
[WITH] [DBA | RESOURCE | CONNECT];
```

对 CREATE USER 语句说明如下:

- 只有系统的超级用户才有权创建一个新的数据库用户。
- 新创建的数据库用户有三种权限:CONNECT、RESOURCE 和 DBA。
- CREATE USER 命令中如果没有指定创建的新用户的权限,默认该用户拥有 CONNECT 权限。拥有 CONNECT 权限的用户不能创建新用户,不能创建模式,也不能创建基本表;只能登录数据库。然后由 DBA 或其他用户授予他应有的权限,根据获得的授权情况他可以对数据库对象进行权限范围内的操作。
- 拥有 RESOURCE 权限的用户能创建基本表和视图,成为所创建对象的属主。但是不能创建模式,不能创建新的用户。数据库对象的属主可以使用 GRANT 语句把该对象上的存取权限授予其他用户。
- 拥有 DBA 权限的用户是系统中的超级用户,可以创建新的用户、创建模式、创建基本表和视图等;DBA 拥有对所有数据库对象的存取权限,还可以把这些权限授予一般用户。

以上说明可以用表 4.6 来总结。

表 4.6 权限与可执行的操作对照表

拥有的权限	可否执行的操作			
	CREATE USER	CREATE SCHEMA	CREATE TABLE	登录数据库,执行数据查询和操纵
DBA	可以	可以	可以	可以
RESOURCE	不可以	不可以	可以	可以
CONNECT	不可以	不可以	不可以	可以,但必须拥有相应权限

一个 DBA 用户可以拥有 CREATE USER、CREATE SCHEMA 和 CREATE TABLE 权限,一个 RESOURCE 用户可以拥有 CREATE TABLE 权限。

有关 CREATE SCHEMA 和 CREATE TABLE 的语句在第 3 章 3.3 中已经讲解了。

4.2.5 数据库角色

数据库角色是被命名的一组与数据库操作相关的权限，角色是权限的集合。因此，可以为一组具有相同权限的用户创建一个角色，使用角色来管理数据库权限可以简化授权的过程。

在 SQL 中首先用 CREATE ROLE 语句创建角色，然后用 GRANT 语句给角色授权。

一、角色的创建

创建角色的 SQL 语句格式是

```
CREATE ROLE <角色名>
```

刚刚创建的角色是空的，没有任何内容。可以用 GRANT 为角色授权。

二、给角色授权

```
GRANT <权限>[,<权限>]…
ON <对象类型>对象名
TO <角色>[,<角色>]…
```

DBA 和用户可以利用 GRANT 语句将权限授予某一个或几个角色。

三、将一个角色授予其他的角色或用户

```
GRANT <角色 1>[,<角色 2>]…
TO <角色 3>[,<用户 1>]…
[WITH ADMIN OPTION]
```

该语句把角色授予某用户，或授予另一个角色。这样，一个角色（例如角色 3）所拥有的权限就是授予它的全部角色（例如角色 1 和角色 2）所包含的权限的总和。

授予者或者是角色的创建者，或者拥有在这个角色上的 ADMIN OPTION。

如果指定了 WITH ADMIN OPTION 子句，则获得某种权限的角色或用户还可以把这种权限再授予其他的角色。

一个角色包含的权限包括直接授予这个角色的全部权限加上其他角色授予这个角色的全部权限。

四、角色权限的收回

```
REVOKE <权限>[,<权限>]…
ON <对象类型> <对象名>
FROM <角色>[,<角色>]…
```

用户可以回收角色的权限，从而修改角色拥有的权限。

REVOKE 动作的执行者或者是角色的创建者，或者拥有在这个（些）角色上的 ADMIN OPTION。

[例 11] 通过角色来实现将一组权限授予一个用户。步骤如下：

1. 首先创建一个角色 R1

```
CREATE ROLE R1;
```

2. 然后使用 GRANT 语句,使角色 R1 拥有 Student 表的 SELECT、UPDATE、INSERT 权限

```
GRANT SELECT,UPDATE,INSERT
ON TABLE Student
TO R1;
```

3. 将这个角色授予王平,张明,赵玲。使他们具有角色 R1 所包含的全部权限

```
GRANT R1
TO 王平,张明,赵玲;
```

4. 当然,也可以一次性的通过 R1 来回收王平的这 3 个权限

```
REVOKE R1
FROM 王平;
```

[例 12] 角色的权限修改

```
GRANT DELETE
ON TABLE Student
TO R1
```

使角色 R1 在原来的基础上增加了 Student 表的 DELETE 权限。

[例 13]

```
REVOKE SELECT
ON TABLE Student
FROM R1;
```

使 R1 减少了 SELECT 权限。

可以看出,通过角色的使用可以使自主授权的执行更加灵活、方便。

4.2.6 强制存取控制(MAC)方法

自主存取控制能够通过授权机制有效地控制对敏感数据的存取。但是由于用户对数据的存取权限是"自主"的,用户可以自由地决定将数据的存取权限授予何人、决定是否也将"授权"的权限授予别人。在这种授权机制下,仍可能存在数据的"无意泄露"。比如,甲将自己权限范围内的某些数据存取权限授权给乙,甲的意图是只允许乙本人操纵这些数据。但甲的这种安全性要求并不能得到保证,因为乙一旦获得了对数据的权限,就可以将数据备份,获得自身权限内的副本,并在不征得甲同意的前提下传播副本。造成这一问题的根本原因就在于,这种机制仅仅通过对数据的存取权限来进行安全控制,而数据本身并无安全性标记。要解决这一问题,就需要对系统控制下的所有主客体实施强制存取控制策略。

所谓 MAC 是指系统为保证更高程度的安全性,按照 TDI/TCSEC 标准中安全策略的要

求，所采取的强制存取检查手段。它不是用户能直接感知或进行控制的。MAC 适用于那些对数据有严格而固定密级分类的部门，例如军事部门或政府部门。

在 MAC 中，DBMS 所管理的全部实体被分为主体和客体两大类。

主体是系统中的活动实体，既包括 DBMS 所管理的实际用户，也包括代表用户的各进程。**客体**是系统中的被动实体，是受主体操纵的，包括文件、基本表、索引、视图等。对于主体和客体，DBMS 为它们每个实例（值）指派一个**敏感度标记**（Label）。

敏感度标记被分成若干级别，例如**绝密**（Top Secret）、**机密**（Secret）、**可信**（Confidential）、**公开**（Public）等。主体的敏感度标记称为**许可证级别**（Clearance Level），客体的敏感度标记称为**密级**（Classification Level）。**MAC 机制**就是通过对比主体的 Label 和客体的 Label，最终确定主体是否能够存取客体。

当某一用户（或某一主体）以标记 label 注册入系统时，系统要求他对任何客体的存取必须遵循如下规则：

（1）仅当主体的许可证级别大于或等于客体的密级时，该主体才能读取相应的客体；

（2）仅当主体的许可证级别等于客体的密级时，该主体才能写相应的客体。

规则（1）的意义是明显的。而规则（2）需要解释一下。在某些系统中，第（2）条规则与这里的规则有些差别。这些系统规定：仅当主体的许可证级别小于或等于客体的密级时，该主体才能写相应的客体，即用户可以为写入的数据对象赋予高于自己的许可证级别的密级。这样一旦数据被写入，该用户自己也不能再读该数据对象了。这两种规则的共同点在于它们均禁止了拥有高许可证级别的主体更新低密级的数据对象，从而防止了敏感数据的泄漏。

强制存取控制（MAC）是对数据本身进行密级标记，无论数据如何复制，标记与数据是一个不可分的整体，只有符合密级标记要求的用户才可以操纵数据，从而提供了更高级别的安全性。

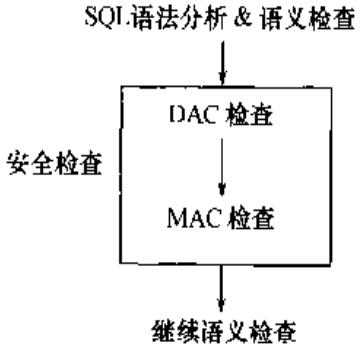


图 4.4 DAC + MAC 安全检查示意图

前面已经提到，较高安全性级别提供的安全保护要包含较低级别的所有保护，因此在实现 MAC 时要首先实现 DAC，即 DAC 与 MAC 共同构成 DBMS 的安全机制，如图 4.4 所示。系统首先进行 DAC 检查，对通过 DAC 检查的允许存取的数据库对象再由系统自动进行 MAC 检查，只有通过 MAC 检查的数据库对象方可存取。

4.3 视图机制

还可以为不同的用户定义不同的视图，把数据对象限制在一定的范围内，也就是说，通过视图机制把要保密的数据对无权存取的用户隐藏起来，从而自动地对数据提供一定程度

的安全保护。

视图机制间接地实现支持存取谓词的用户权限定义。例如,在某大学中假定王平老师只能检索计算机系学生的信息,系主任张明具有检索和增删改计算机系学生信息的所有权限。这就要求系统能支持"存取谓词"的用户权限定义。在不直接支持存取谓词的系统中,可以先建立计算机系学生的视图 CS_Student,然后在视图上进一步定义存取权限。

[例 14] 建立计算机系学生的视图,把对该视图的 SELECT 权限授于王平,把该视图上的所有操作权限授于张明

```
CREATE VIEW CS_Student              /* 先建立视图 CS_Student */
AS
SELECT *
FROM Student
WHERE Sdept ='CS';

GRANT SELECT
ON CS_Student
TO 王平;

GRANT ALL PRIVILIGES
ON CS_Student
TO 张明;
```

4.4 审计(Audit)

前面讲的用户标识与鉴别、存取控制仅是安全性标准的一个重要方面(安全策略方面)不是全部。为了使 DBMS 达到一定的安全级别,还需要在其他方面提供相应的支持。例如按照 TDI/TCSEC 标准中安全策略的要求,"审计"功能就是 DBMS 达到 C2 以上安全级别必不可少的一项指标。

因为任何系统的安全保护措施都不是完美无缺的,蓄意盗窃、破坏数据的人总是想方设法打破控制。审计功能把用户对数据库的所有操作自动记录下来放入审计日志(Audit Log)中。DBA 可以利用审计跟踪的信息,重现导致数据库现有状况的一系列事件,找出非法存取数据的人、时间和内容等。

审计通常是很费时间和空间的,所以 DBMS 往往都将其作为可选特征,允许 DBA 根据应用对安全性的要求,灵活地打开或关闭审计功能。审计功能一般主要用于安全性要求较高的部门。

审计一般可以分为用户级审计和系统级审计。用户级审计是任何用户可设置的审计,主要是用户针对自己创建的数据库表或视图进行审计,记录所有用户对这些表或视图的一

切成功和(或)不成功的访问要求以及各种类型的 SQL 操作。

系统级审计只能由 DBA 设置,用以监测成功或失败的登录要求、监测 GRANT 和 REVOKE 操作以及其他数据库级权限下的操作。

AUDIT 语句用来设置审计功能,NOAUDIT 语句取消审计功能。

[例 15] 对修改 SC 表结构或修改 SC 表数据的操作进行审计。

```
AUDIT ALTER,UPDATE
ON SC;
```

[例 16] 取消对 SC 表的一切审计。

```
NOAUDIT ALTER,UPDATE
ON SC;
```

审计设置以及审计内容一般都存放在数据字典中。必须把审计开关打开(即把系统参数 audit_trail 设为 true),才可以在系统表(SYS_AUDITTRAIL)中查看审计信息。

4.5 数据加密

对于高度敏感性数据,例如财务数据、军事数据、国家机密,除以上安全性措施外,还可以采用数据加密技术。

数据加密是防止数据库中数据在存储和传输中失密的有效手段。加密的基本思想是根据一定的算法将原始数据(术语为**明文**,Plain text)变换为不可直接识别的格式(术语为**密文**,Cipher text),从而使得不知道解密算法的人无法获知数据的内容。

加密方法主要有两种,一种是替换方法,该方法使用**密钥**(Encryption Key)将明文中的每一个字符转换为密文中的一个字符。另一种是置换方法,该方法仅将明文的字符按不同的顺序重新排列。单独使用这两种方法的任意一种都是不够安全的。但是将这两种方法结合起来就能提供相当高的安全程度。采用这种结合算法的例子是美国 1977 年制定的官方加密标准,数据加密标准(Data Encryption Standard,简称 DES)。

有关 DES 密钥加密技术及密钥管理问题等已超出本书范围,这里不再讨论。

目前有些数据库产品提供了数据加密例行程序,可根据用户的要求自动对存储和传输的数据进行加密处理。另一些数据库产品虽然本身未提供加密程序,但提供了接口,允许用户用其他厂商的加密程序对数据加密。

由于数据加密与解密也是比较费时的操作,而且数据加密与解密程序会占用大量系统资源,因此数据加密功能通常也作为可选特征,允许用户自由选择,只对高度机密的数据加密。

4.6 统计数据库安全性

一般地,统计数据库允许用户查询聚集类型的信息(例如合计、平均值等),但是不允许查询单个记录信息。例如,查询"程序员的平均工资是多少?"是合法的,但是查询"程序员张勇的工资是多少?"就不允许。

在统计数据库中存在着特殊的安全性问题,即可能存在着隐蔽的信息通道,使得可以从合法的查询中推导出不合法的信息。例如下面两个查询都是合法的:

- 本公司共有多少女高级程序员?
- 本公司女高级程序员的工资总额是多少?

如果第1个查询的结果是"1",那么第2个查询的结果显然就是这个程序员的工资数。这样统计数据库的安全性机制就失效了。为了解决这个问题,可以规定任何查询至少要涉及 N 个以上的记录(N 足够大)。但是即使这样,还是存在另外的泄密途径,看下面的例子:

某个用户A想知道另一用户B的工资数额,他可以通过下列两个合法查询获取:

- 用户A和其他 N 个程序员的工资总额是多少?
- 用户B和其他 N 个程序员的工资总额是多少?

假设第一个查询的结果是X,第二个查询的结果是Y,由于用户A知道自己的工资是Z,那么他可以计算出用户B的工资 = Y - (X - Z)。

这个例子的关键之处在于两个查询之间有很多重复的数据项(即其他 N 个程序员的工资)。因此可以再规定任意两个查询的相交数据项不能超过 M 个。这样就使得获取他人的数据更加困难了。可以证明,在上述两条规定下,如果想获知用户B的工资额,用户A至少需要进行 $1+(N-2)/M$ 次查询。

当然可以继续规定任一用户的查询次数不能超过 $1+(N-2)/M$,但是如果两个用户合作查询就可以使这一规定仍然失效。

另外还有其他一些方法用于解决统计数据库的安全性问题,例如数据污染。但是无论采用什么安全性机制,都仍然会存在绕过这些机制的途径。好的安全性措施应该使得那些试图破坏安全的人所花费的代价远远超过他们所得到的利益,这也是整个数据库安全机制设计的目标。

4.7 小 结

随着计算机特别是计算机网络的发展,数据的共享日益加强,数据的安全保密越来越重要。DBMS是管理数据的核心,因而其自身必须具有一整套完整而有效的安全性机制。

计算机以及信息安全技术方面有一系列的安全标准,最有影响的当推TCSEC和CC这

两个标准。目前 CC 已经基本取代了 TCSEC,成为评估信息产品安全性的主要标准。

实现数据库系统安全性的技术和方法有多种,最重要的是存取控制技术、视图技术和审计技术。自主存取控制功能一般是通过 SQL 的 GRANT 语句和 REVOKE 语句来实现的。对数据库模式的授权则由 DBA 在创建用户时通过 CREATE USER 语句来实现。数据库角色是一组权限的集合。使用角色来管理数据库权限可以简化授权的过程。在 SQL 中用 CREATE ROLE 语句创建角色,用 GRANT 语句给角色授权。

表 4.3 关系数据库系统中的存取权限中,列权限包括 SELECT,REFERENCES,INSERT,UPDATE,其含义与表权限类似。需要说明的是对列的 UPDATE 权限,指对于表中存在的某一列的值可以进行修改。当然,有了这个权限之后,在修改的过程中还要遵守表在创建的时候定义的主码及其他约束。列上的 INSERT 权限指,用户可以插入一个元组。对于插入的元组,授予了用户 INSERT 权限的列,用户可以插入指定的值,其他列或者为空,或者为默认值。在给用户授予列 INSERT 权限的时候,一定要包含主码的 INSERT 权限,否则用户的插入动作会因为主码为空而被拒绝。

习 题

1. 什么是数据库的安全性?
2. 数据库安全性和计算机系统的安全性有什么关系?
3. 试述信息安全标准的发展历史,试述 TDI/TCSEC 和 CC V2.1 标准的基本内容。
4. 试述 CC 评估保证级划分的基本内容。
5. 试述实现数据库安全性控制的常用方法和技术。
6. 什么是数据库中的自主存取控制方法和强制存取控制方法?
7. SQL 中提供了哪些数据控制(自主存取控制)的语句?请试举几例说明它们的使用方法。
8. 今有两个关系模式:

 职工(职工号,姓名,年龄,职务,工资,部门号)

 部门(部门号,名称,经理名,地址,电话号)

请用 SQL 的 GRANT 和 REVOKE 语句(加上视图机制)完成以下授权定义或存取控制功能:

(a) 用户王明对两个表有 SELECT 权力;

(b) 用户李勇对两个表有 INSERT 和 DELETE 权力;

(c) 每个职工只对自己的记录有 SELECT 权力;

(d) 用户刘星对职工表有 SELECT 权力,对工资字段具有更新权力;

(e) 用户张新具有修改这两个表的结构的权力;

(f) 用户周平具有对两个表所有权力(读,插,改,删数据),并具有给其他用户授权的权力;

(g) 用户杨兰具有从每个部门职工中 SELECT 最高工资,最低工资,平均工资的权力,他不能查看每个人的工资。

9. 把习题 8 中(a)~(g)的每一种情况,撤销各用户所授予的权力。

10. 为什么强制存取控制提供了更高级别的数据库安全性?

11. 理解并解释 MAC 机制中主体、客体、敏感度标记的含义。

12. 举例说明 MAC 机制如何确定主体能否存取客体的。

13. 什么是数据库的审计功能,为什么要提供审计功能?

14. 统计数据库中存在何种特殊的安全性问题?

*15. 试述你了解的某一个实际的 DBMS 产品的安全性措施。

实验 3　数据控制(安全性部分)

一、实验目的

熟悉通过 SQL 对数据进行安全性控制。完成作业的上机练习。

二、实验平台

在实验 1 中安装的 RDBMS 及其交互查询工具。

三、实验内容和要求

使用 SQL 对数据进行安全性控制,包括:授权和权力回收。操作完成后看看已授权的用户是否真正具有授予的数据操作的权力了;权力收回操作之后的用户是否确实丧失了收回的数据操作的权力)。根据以下要求认真填写实验报告,记录所有的实验用例。

四、实验报告

《数据库系统概论》实验报告				
题目:	姓名		日期	
实验环境:				
实验内容与完成情况:				
出现的问题:				
解决方案(列出遇到的问题和解决办法,列出没有解决的问题):				

本章参考文献

1 Department of Defense(USA). Department of Defense Trusted Computer System Evaluation Criteria (Orange Book), Rainbow Series-12/85 (DoD 5200.28-std)

2 Evaluated Products List Indexed by Rating. www.radium.ncsc/tpep/library/rainbow

3 Department of Defense. Trusted Database Management System of TCSEC, 1991

4 Fernfandez E B, Summers R C, Wood C. Database Security and Integrity. Reading Mass, Addison-Wesley, 1981
(本文献详细地讨论了数据库的安全性和完整性问题。)

5 Denning D E. Cryptography and Data Security, Reading Mass, Addison Wesley, 1983

6 Zloof M M. Security and Integrity Within the Query-By-Example Data Base Management Language. IBM Research Report RC 6982, 1978

7 Jajodia Sushil, Sandhu Ravi. Toward a Multilevel Secure Relation Data Model. Proc. ACM SIGMOD International Conference on Management Data, 1991

8 Griffiths P, Wade B. An Authorization Mechanism for a Relational Database System. ACM TODS, 1:3, 1976

（本文讨论了 SYSTEM R 的授权机制。关系数据库系统原型 SYSTEM R 首先提出了授权和撤销权力的概念。）

9 Leiss E. Principles of Data Security, Plenum Press, 1982

（本文献详细地讨论了计算机系统和数据库的安全性问题。）

10 Chin F. Security in Statistical Databases for Queries with Small Counts. ACM TODS, 3:1, 1978

（本文讨论了统计数据库的安全性和保护技术。）

11 Chin F, Ozsoyoglu G. Statistical Database Design. ACM TODS, 6:1, 1981

12 Leiss E, Randomizing A. A Practical Method for Protecting Statistical Database Against Compromise. in Proceedings of VLDB, 1982

13 刘启原. 数据库与信息系统的安全. 数据库技术丛书之一, 北京: 科学出版社, 1999

（本书系统讲解了信息系统的安全性问题，包括安全模型、与安全有关的标准、密码学与信息保密、攻击检测技术和理论、数据库的安全性、数据仓库的安全问题、网络安全问题、Java 语言及其安全性问题、信息安全和病毒问题等。）

14 张孝. 可信 COBASE 的系统强制存取控制的设计与实现. 中国人民大学硕士论文, 1998

（本文系统讨论了数据库系统的安全性问题，给出了可信 COBASE 的系统设计和实现技术。可信 COBASE 是 COBASE V2.0 的安全版本。COBASE V2.0 是由北京大学、中国人民大学、中国软件总公司研制的 RDBMS。可信 COBASE 依据 TCSEC 和 TDI 对 B1 级的要求进行设计，实现了强制存取控制（MAC）和安全审计。）

15 文继荣、张孝、罗立等. 可信 COBASE 的设计策略：数据强制存取控制机制及其实现. 第 14 届全国数据库学术会议论文集, 1997

16 张俊, 彭朝晖, 肖艳芹, 王珊. DBMS 安全性评估保护轮廓 PP 的研究与开发. 第 22 届全国数据库学术会议论文集, 2005

（本文在深入研究 CC 以及国外主流 DBMS CC 安全性评估情况的基础上，总结出 DBMS PP 的开发原则和方法，并初步研究和开发了一个 EAL4 级 DBMS PP。我们根据该 PP 对人大金仓 DBMS KingbaseES 进行安全性开发、测试和内部评估，取得良好的效果。）

17 Common Criteria for Information Technology Security Evaluation. Version 2.1. ISO/IEC 15408. CCIB-99-031, 1999.

18 国家技术监督局. 信息技术、安全技术、信息技术安全性评估准则. GB/T 18336-2001, 2001

19 NIST. Development Process For US Government Protection Profiles (PP). V3.0, 2004

20 NIST. U. S. Government Protection Profile for Database Management Systems in Basic Robustness Environments Version 1.0 , 2004

21 NIST. Consistency Instruction Manual For development of US Government Protection Profiles For use in Medium Robustness Environments. Release 3.0, 2005

22 NIST. Consistency Instruction Manual For development of US Government Protection Profiles For use in Basic Robustness Environments. Release 3.0, 2005

第五章 数据库完整性

数据库的完整性是指数据的正确性和相容性。

例如,学生的学号必须唯一;性别只能是男或女;本科学生年龄的取值范围为 14 ~ 50 的整数;学生所选的课程必须是学校开设的课程,学生所在的院系必须是学校已成立的院系等。

数据的完整性和安全性是两个不同的概念。

数据的完整性是为了防止数据库中存在不符合语义的数据,也就是防止数据库中存在不正确的数据。数据的安全性是保护数据库防止恶意的破坏和非法的存取。因此,完整性检查和控制的防范对象是不合语义的、不正确的数据,防止它们进入数据库。安全性控制的防范对象是非法用户和非法操作,防止他们对数据库数据的非法存取。

为维护数据库的完整性,DBMS 必须能够:

1. 提供定义完整性约束条件的机制

完整性约束条件也称为完整性规则,是数据库中的数据必须满足的语义约束条件。SQL 标准使用了一系列概念来描述完整性,包括关系模型的实体完整性、参照完整性和用户定义完整性。这些完整性一般由 SQL 的 DDL 语句来实现。它们作为数据库模式的一部分存入数据字典中。

2. 提供完整性检查的方法

DBMS 中检查数据是否满足完整性约束条件的机制称为完整性检查。一般在 INSERT、UPDATE、DELETE 语句执行后开始检查,也可以在事务提交时检查。检查这些操作执行后数据库中的数据是否违背了完整性约束条件。

3. 违约处理

DBMS 若发现用户的操作违背了完整性约束条件,就采取一定的动作,如拒绝(NO ACTION)执行该操作,或级连(CASCADE)执行其他操作,进行违约处理以保证数据的完整性。

目前商用的 DBMS 产品都支持完整性控制。即完整性定义和检查控制由 DBMS 实现,不必由应用程序来完成,从而减轻了应用程序员的负担。

更重要的是使得完整性控制成为 DBMS 核心支持的功能,从而能够为所有的用户和所有的应用提供一致的数据库完整性。因为由应用程序来实现完整性控制是有漏洞的。有的

应用程序定义的完整性约束条件可能被其他应用程序破坏，数据库数据的正确性仍然无法保障。

我们在第2章2.3已经讲解了关系数据库三类完整性约束的基本概念，下面讲解SQL语言中实现这些完整性控制功能的方法。

5.1 实体完整性

5.1.1 实体完整性定义

关系模型的实体完整性在CREATE TABLE中用PRIMARY KEY定义。对单属性构成的码有两种说明方法，一种是定义为列级约束条件，另一种是定义为表级约束条件。对多个属性构成的码只有一种说明方法，即定义为表级约束条件。

[例1] 将Student表中的Sno属性定义为码。

```
CREATE TABLE Student
    (Sno CHAR(9) PRIMARY KEY,                    /* 在列级定义主码 */
     Sname CHAR(20) NOT NULL,
     Ssex CHAR(2) ,
     Sage SMALLINT,
     Sdept CHAR(20)
     );
```

或者

```
CREATE TABLE Student
    (Sno CHAR(9),
     Sname CHAR(20) NOT NULL,
     Ssex CHAR(2) ,
     Sage SMALLINT,
     Sdept CHAR(20),
     PRIMARY KEY (Sno)                           /* 在表级定义主码 */
     );
```

[例2] 将SC表中的Sno,Cno属性组定义为码

```
CREATE TABLE SC
    (Sno CHAR(9) NOT NULL,
     Cno CHAR(4) NOT NULL,
     Grade SMALLINT,
     PRIMARY KEY (Sno,Cno)                       /* 只能在表级定义主码 */
    );
```

5.1.2 实体完整性检查和违约处理

用 PRIMARY KEY 短语定义了关系的主码后，每当用户程序对基本表插入一条记录或者对主码列进行更新操作时，RDBMS 将按照第二章 2.3.2 小节中讲解的实体完整性规则自动进行检查。包括：

1. 检查主码值是否唯一，如果不唯一则拒绝插入或修改。
2. 检查主码的各个属性是否为空，只要有一个为空就拒绝插入或修改。

从而保证了实体完整性。

检查记录中主码值是否唯一的一种方法是进行全表扫描。依次判断表中每一条记录的主码值与将插入记录上的主码值(或者修改的新主码值)是否相同，如图 5.1 所示。

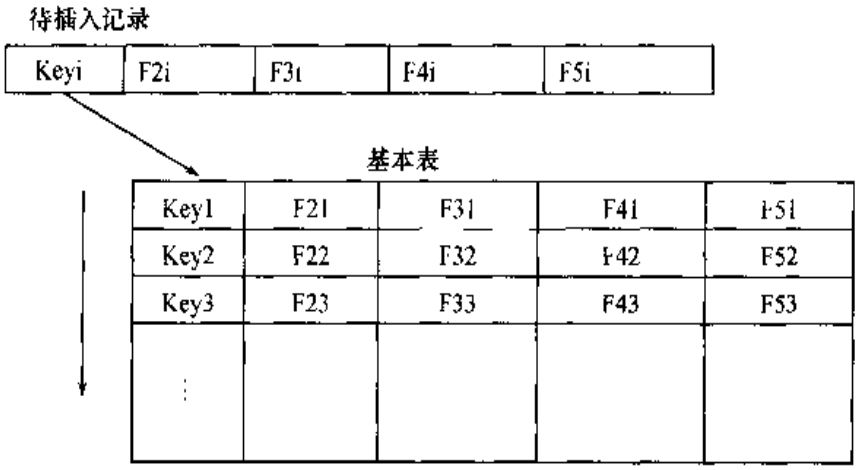


图 5.1 用全表扫描方法检查主码唯一性

全表扫描是十分耗时的。为了避免对基本表进行全表扫描，RDBMS 核心一般都在主码上自动建立一个索引，如图 5.2 的 B + 树索引。通过索引查找基本表中是否已经存在新的

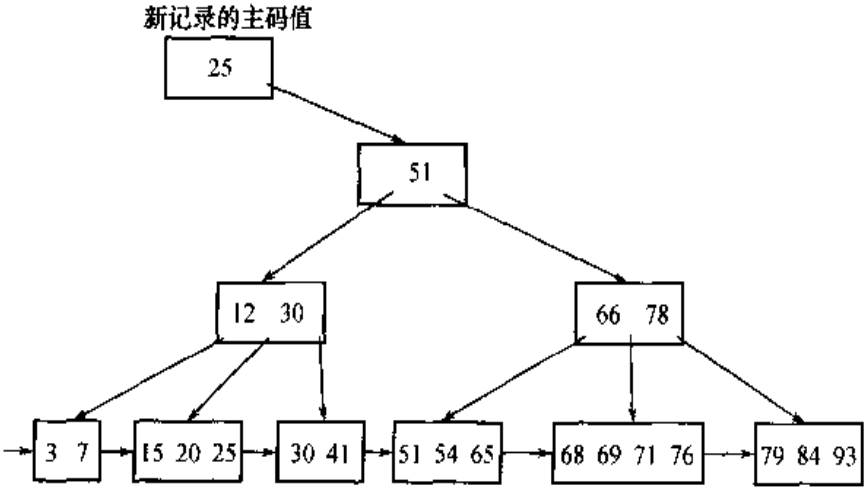


图 5.2 使用索引检查主码唯一性

主码值，将大大提高效率。例如，如果新插入记录的主码值是25，通过主码索引，从B+树的根结点开始查找，只要读取3个结点就可以知道该主码值已经存在，所以不能插入这条记录。这3个结点是根结点(51)、中间结点(12 30)、叶结点(15 20 25)。如果新插入记录的主码值是86，也只要查找3个结点就可以知道该主码值不存在，所以可以插入该记录。

5.2　参照完整性

5.2.1　参照完整性定义

关系模型的参照完整性在CREATE TABLE中用FOREIGN KEY短语定义哪些列为外码，用REFERENCES短语指明这些外码参照哪些表的主码。

例如，关系SC中一个元组表示一个学生选修的某门课程的成绩，(Sno,Cno)是主码。Sno,Cno分别参照引用Student表的主码和Course表的主码。

[例3]　定义SC中的参照完整性

```
CREATE TABLE SC
  (Sno CHAR(9) NOT NULL,
   Cno CHAR(4) NOT NULL,
   Grade SMALLINT,
   PRIMARY KEY (Sno, Cno),                          /* 在表级定义实体完整性 */
   FOREIGN KEY (Sno) REFERENCES Student(Sno),       /* 在表级定义参照完整性 */
   FOREIGN KEY (Cno) REFERENCES Course(Cno)         /* 在表级定义参照完整性 */
  );
```

5.2.2　参照完整性检查和违约处理

一个参照完整性将两个表中的相应元组联系起来了。因此，对被参照表和参照表进行增删改操作时有可能破坏参照完整性，必须进行检查。

例如，对表SC和Student有四种可能破坏参照完整性的情况，如表5.1所示。

1. SC表中增加一个元组，该元组的Sno属性的值在表Student中找不到一个元组，其Sno属性的值与之相等。

2. 修改SC表中的一个元组，修改后该元组的Sno属性的值在表Student中找不到一个元组，其Sno属性的值与之相等。

3. 从Student表中删除一个元组，造成SC表中某些元组的Sno属性的值在表Student中找不到一个元组，其Sno属性的值与之相等。

4. 修改Student表中一个元组的Sno属性，造成SC表中某些元组的Sno属性的值在表Student中找不到一个元组，其Sno属性的值与之相等。

表 5.1 可能破坏参照完整性的情况及违约处理

被参照表（例如 Student）		参照表（例如 SC）	违约处理
可能破坏参照完整性	←	插入元组	拒绝
可能破坏参照完整性	←	修改外码值	拒绝
删除元组	→	可能破坏参照完整性	拒绝/级连删除/设置为空值
修改主码值	→	可能破坏参照完整性	拒绝/级连修改/设置为空值

当上述的不一致发生时，系统可以采用以下的策略加以处理。

1. 拒绝（NO ACTION）执行

不允许该操作执行。该策略一般设置为默认策略。

2. 级连（CASCADE）操作

当删除或修改被参照表（Student）的一个元组造成了与参照表（SC）的不一致，则删除或修改参照表中的所有造成不一致的元组。

例如，删除 Student 表中的元组，Sno 值为 200215121，则从要 SC 表中级连删除 SC. Sno = '200215121'的所有元组）。

3. 设置为空值（SET - NULL）

当删除或修改被参照表的一个元组时造成了不一致，则将参照表中的所有造成不一致的元组的对应属性设置为空值。例如，有下面 2 个关系（本书第二章 2.3.3 小节的[例 1]）

学生（学号，姓名，性别，专业号，年龄）

专业（专业号，专业名）

学生关系的“专业号”是外码，因为专业号是专业关系的主码。

假设专业表中某个元组被删除，专业号为 12，按照设置为空值的策略，就要把学生表中专业号 = 12 的所有元组的专业号设置为空值。这对应了这样的语义：某个专业删除了，该专业的所有学生专业未定，等待重新分配专业。

这里要讲解一下外码能否接受空值的问题。

例如，这里的学生表中，“专业号”是外码，按照应用的实际情况可以取空值，表示这个学生的专业尚未确定。

但在学生 - 选课数据库中，Student 关系为被参照关系，其主码为 Sno。SC 为参照关系，Sno 为外码。若 SC 的 Sno 为空值，则表明尚不存在的某个学生，或者某个不知学号的学生，选修了某门课程，其成绩记录在 Grade 列中。这与学校的应用环境是不相符的，因此 SC 的 Sno 列不能取空值。同样，SC 的 Cno 列不能取空值。

因此对于参照完整性时，除了应该定义外码，还应定义外码列是否允许空值。

一般地，当对参照表和被参照表的操作违反了参照完整性，系统选用默认策略，即拒绝执行。如果想让系统采用其他的策略则必须在创建表的时候显式地加以说明。

[例 4] 显式说明参照完整性的违约处理示例。

```
CREATE TABLE SC
(Sno CHAR(9) NOT NULL,
 Cno CHAR(4) NOT NULL,
 Grade SMALLINT,
 PRIMARY KEY(Sno,Cno),                                    /* 在表级定义实体完整性 */
 FOREIGN KEY (Sno) REFERENCES Student(Sno)                 /* 在表级定义参照完整性 */
   ON DELETE CASCADE    /* 当删除 student 表中的元组时,级连删除 SC 表中相应的元组 */
   ON UPDATE CASCADE,   /* 当更新 student 表中的 sno 时,级连更新 SC 表中相应的元组 */
 FOREIGN KEY (Cno) REFERENCES Course(Cno)                  /* 在表级定义参照完整性 */
   ON DELETE NO ACTION  /* 当删除 course 表中的元组造成了与 SC 表不一致时拒绝删除 */
   ON UPDATE CASCADE    /* 当更新 course 表中的 cno 时,级连更新 SC 表中相应的元组 */
);
```

可以对 DELETE 和 UPDATE 采用不同的策略。例如[例 4]中当删除被参照表 course 表中的元组,造成了与参照表(SC 表)不一致时,拒绝删除被参照表的元组;对更新操作则采取级连更新的策略。

从上面的讨论看到 RDBMS 在实现参照完整性时,除了要提供定义主码、外码的机制外,还需要提供不同的策略供用户选择。选择哪种策略,要根据应用环境的要求确定。

5.3 用户定义的完整性

在第二章 2.3.4 小节中已经讲解了什么是用户定义的完整性。用户定义的完整性就是针对某一具体应用的数据必须满足的语义要求。

目前的 RDBMS 都提供了定义和检验这类完整性的机制,使用了和实体完整性、参照完整性相同的技术和方法来处理它们,而不必由应用程序承担这一功能。

5.3.1 属性上的约束条件的定义

在 CREATE TABLE 中定义属性的同时可以根据应用要求,定义属性上的约束条件,即属性值限制,包括:

- 列值非空(NOT NULL 短语)
- 列值唯一(UNIQUE 短语)
- 检查列值是否满足一个布尔表达式(CHECK 短语)

1. 不允许取空值

[例 5] 在定义 SC 表时,说明 Sno、Cno、Grade 属性不允许取空值。

```
CREATE TABLE SC
    (Sno CHAR(9) NOT NULL,     /* Sno 属性不允许取空值 */
```

```
    Cno CHAR(4) NOT NULL,     /* Cno 属性不允许取空值 */
    Grade SMALLINT NOT NULL,  /* Grade 属性不允许取空值 */
    PRIMARY KEY (Sno, Cno),   /* 如果在表级定义实体完整性,隐含了 Sno, Cno 不允许
                                 取空值则在列级不允许取空值的定义就不必写了 */
                    ⋮
);
```

2. 列值唯一

[例 6] 建立部门表 DEPT,要求部门名称 Dname 列取值唯一,部门编号 Deptno 列为主码。

```
CREATE TABLE DEPT
 (Deptno NUMERIC(2),
  Dname CHAR(9) UNIQUE,        /* 要求 Dname 列值唯一 */
  Location CHAR(10),
  PRIMARY KEY (Deptno)
 );
```

3. 用 CHECK 短语指定列值应该满足的条件

[例 7] Student 表的 Ssex 只允许取“男”或“女”。

```
CREATE TABLE Student
(Sno CHAR(9) PRIMARY KEY,                    /* 在列级定义主码 */
Sname CHAR(8) NOT NULL,                      /* Sname 属性不允许取空值 */
Ssex CHAR(2) CHECK (Ssex IN ('男','女')),    /* 性别属性 Ssex 只允许取'男'或'女' */
Sage SMALLINT,
Sdept CHAR(20)
);
```

[例 8] SC 表的 Grade 的值应该在 0 和 100 之间。

```
CREATE TABLE SC
 (Sno CHAR(9) NOT NULL,
  Cno CHAR(4) NOT NULL,
  Grade SMALLINT CHECK (Grade >=0 AND Grade <=100),
  PRIMARY KEY (Sno,Cno),
  FOREIGN KEY (Sno) REFERENCES Student(Sno),
  FOREIGN KEY (Cno) REFERENCES Course(Cno)
 );
```

5.3.2 属性上的约束条件检查和违约处理

当往表中插入元组或修改属性的值时,RDBMS 就检查属性上的约束条件是否被满足,如果不满足则操作被拒绝执行。

5.3.3 元组上的约束条件的定义

与属性上约束条件的定义类似,在 CREATE TABLE 语句中可以用 CHECK 短语定义元组上的约束条件,即元组级的限制。同属性值限制相比,元组级的限制可以设置不同属性之间的取值的相互约束条件。

[例 9] 当学生的性别是男时,其名字不能以 Ms. 打头。

```
CREATE TABLE Student
  (Sno CHAR(9),
   Sname CHAR(8) NOT NULL,
   Ssex CHAR(2),
   Sage SMALLINT,
   Sdept CHAR(20),
   PRIMARY KEY (Sno),
   CHECK (Ssex = '女' OR Sname NOT LIKE 'Ms. %')
  );          /* 定义了元组中 Sname 和 Ssex 两个属性值之间的约束条件 */
```

性别是女性的元组都能通过该项检查,因为 Ssex = '女'成立;当性别是男性时,要通过检查则名字一定不能以 Ms. 打头,因为 Ssex = '男'时,条件要想为真值,Sname NOT LIKE 'Ms. %' 必须为真值。

5.3.4 元组上的约束条件检查和违约处理

当往表中插入元组或修改属性的值时,RDBMS 就检查元组上的约束条件是否被满足,如果不满足则操作被拒绝执行。

5.4 完整性约束命名子句

以上讲解的完整性约束条件都在 CREATE TABLE 语句中定义。SQL 还在 CREATE TABLE 语句中提供了完整性约束命名子句 CONSTRAINT,用来对完整性约束条件命名。从而可以灵活地增加、删除一个完整性约束条件。

1. 完整性约束命名子句

CONSTRAINT <完整性约束条件名>[PRIMARY KEY 短语 | FOREIGN KEY 短语 | CHECK 短语]

[例 10] 建立学生登记表 Student,要求学号在 90 000 ~ 99 999 之间,姓名不能取空值,年龄小于 30,性别只能是"男"或"女"。

```
CREATE TABLE Student
(Sno NUMERIC(6)
```

```
    CONSTRAINT C1 CHECK (Sno BETWEEN 90000 AND 99999),
  Sname CHAR(20)
    CONSTRAINT C2 NOT NULL,
  Sage NUMERIC(3)
    CONSTRAINT C3 CHECK (Sage < 30),
  Ssex CHAR(2)
    CONSTRAINT C4 CHECK (Ssex IN ('男','女')),
  CONSTRAINT StudentKey PRIMARY KEY(Sno)
 );
```

在 Student 表上建立了 5 个约束条件,包括主码约束(命名为 StudentKey)以及 C1、C2、C3、C4 四个列级约束。

[例 11] 建立教师表 TEACHER,要求每个教师的应发工资不低于 3 000 元。

应发工资实际上就是实发工资列 Sal 与扣除项 Deduct 之和。

```
CREATE TABLE TEACHER
  (Eno NUMERIC(4) PRIMARY KEY,                    /* 在列级定义主码 */
   Ename CHAR(10),
   Job CHAR(8),
   Sal NUMERIC(7,2),
   Deduct NUMERIC(7,2),
   Deptno NUMERIC(2),
   CONSTRAINT EMPFKey FOREIGN KEY (Deptno) REFERENCES DEPT(Deptno),
   CONSTRAINTS C1 CHECK (Sal + Deduct >= 3000)
  );
```

2. 修改表中的完整性限制

我们可以使用 ALTER TABLE 语句修改表中的完整性限制。

[例 12] 去掉[例 10]student 表中对性别的限制。

```
ALTER TABLE Student
  DROP CONSTRAINT C4;
```

[例 13] 修改表 Student 中的约束条件,要求学号改为在 900 000 ~ 999 999 之间,年龄由小于 30 改为小于 40。

可以先删除原来的约束条件,再增加新的约束条件。

```
ALTER TABLE Student
  DROP CONSTRAINT C1;
ALTER TABLE Student
  ADD CONSTRAINT C1 CHECK (Sno BETWEEN 900 000 AND 999 999),
ALTER TABLE Student
  DROP CONSTRAINT C3;
ALTER TABLE Student
```

```
ADD CONSTRAINT C3 CHECK (Sage < 40);
```

*5.5 域中的完整性限制

SQL 支持域的概念,并可以用 CREATE DOMAIN 语句建立一个域以及该域应该满足的完整性约束条件。

[例 14] 建立一个性别域,并声明性别域的取值范围

```
CREATE DOMAIN GenderDomain CHAR(2)
    CHECK (VALUE IN ('男','女'));
```

这样[例 10]中对 Ssex 的说明可以改写为

```
Ssex GenderDomain
```

[例 15] 建立一个性别域 GenderDomain,并对其中的限制命名。

```
CREATE DOMAIN GenderDomain CHAR(2)
    CONSTRAINT GD CHECK ( VALUE IN ('男','女') );
```

[例 16] 删除域 GenderDomain 的限制条件 GD。

```
ALTER DOMAIN GenderDomain
    DROP CONSTRAINT GD;
```

[例 17] 在域 GenderDomain 上增加限制条件 GDD。

```
ALTER DOMAIN GenderDomain
    ADD CONSTRAINT GDD CHECK (VALUE IN ('1','0'));
```

这样,通过[例 16]和[例 17],就把性别的取值范围由('男','女')改为('1','0')。

5.6 触 发 器

触发器(Trigger)是用户定义在关系表上的一类由事件驱动的特殊过程。一旦定义,任何用户对表的增、删、改操作均由服务器自动激活相应的触发器,在 DBMS 核心层进行集中的完整性控制。触发器类似于约束,但是比约束更加灵活,可以实施比 FOREIGN KEY 约束、CHECK 约束更为复杂的检查和操作,具有更精细和更强大的数据控制能力。

触发器并不是 SQL92 或 SQL99 核心 SQL 规范的内容,但是很多 RDBMS 很早就支持触发器,因此不同的 RDBMS 实现的触发器语法也会有所不同。请读者在上机实验时注意阅读实验系统的使用说明。

5.6.1 定义触发器

SQL 使用 CREATE TRIGGER 命令建立触发器,其一般格式为

```
CREATE TRIGGER <触发器名>
  {BEFORE | AFTER} <触发事件> ON <表名>
  FOR EACH {ROW | STATEMENT}
  [WHEN <触发条件>]
  <触发动作体>
```

下面对定义触发器的各部分语法进行详细说明。

1. 表的拥有者即创建表的用户才可以在表上创建触发器,并且一个表上只能创建一定数量的触发器。

2. 触发器名

触发器名可以包含模式名,也可以不包含模式名。同一模式下,触发器名必须是唯一的;并且触发器名和<表名>必须在同一模式下。

3. 表名

当这个表的数据发生变化时,将激活定义在该表上相应<触发事件>的触发器,因此,该表也称为触发器的目标表。

4. 触发事件

触发事件可以是 INSERT、DELETE 或 UPDATE,也可以是这几个事件的组合,如 INSERT OR DELETE 等。UPDATE 后面还可以有 OF <触发列,...>,即进一步指明修改哪些列时触发器激活。

5. 触发器类型

触发器按照所触发动作的间隔尺寸可以分为行级触发器(FOR EACH ROW)和语句级触发器(FOR EACH STATEMENT)。

例如,假设在[例 11]的 TEACHER 表上创建了一个 AFTER UPDATE 触发器。如果表 TEACHER 有 1 000 行,执行如下语句:

```
UPDATE TEACHER SET Deptnoe = 5;
```

如果该触发器为语句级触发器,那么执行完该语句后,触发动作只发生一次。如果是行级触发器,触发动作将执行 1 000 次。

6. 触发条件

触发器被激活时,只有当触发条件为真时触发动作体才执行;否则触发动作体不执行。如果省略 WHEN 触发条件,则触发动作体在触发器激活后立即执行。

7. 触发动作体

触发动作体既可以是一个匿名 PL/SQL 过程块,也可以是对已创建存储过程的调用。如果是行级触发器,在两种情况下,用户都可以在过程体中使用 NEW 和 OLD 引用 UPDATE/INSERT 事件之后的新值和 UPDATE/DELETE 事件之前的旧值。如果是语句级触发器,则不能在触发动作体中使用 NEW 或 OLD 进行引用。

如果触发动作体执行失败,激活触发器的事件就会终止执行,触发器的目标表或触发器

可能影响的其他对象不发生任何变化。

[例 18] 定义一个 BEFORE 行级触发器，为教师表 Teacher 定义完整性规则“教授的工资不得低于 4 000 元，如果低于 4 000 元，自动改为 4 000 元”。

```
CREATE TRIGGER Insert_Or_Update_Sal          /* 在教师表 Teacher 上定义触发器 */
  BEFORE INSERT OR UPDATE ON Teacher         /* 触发事件是插入或更新操作 */
  FOR EACH ROW                               /* 这是行级触发器 */
  AS BEGIN                                   /* 定义触发动作体，这是一个 PL/SQL 过程块 */
    IF (new.pJob ='教授') AND (new.Sal < 4000) THEN     /* 因为是行级触发器，可在 */
        new.Sal := 4000;                     /* 过程体中使用插入或更新操作后的新值 */
    END IF;
  END;                                       /* 触发动作体结束 */
```

[例 19] 定义 AFTER 行级触发器，当教师表 Teacher 的工资发生变化后就自动在工资变化表 Sal_log 中增加一条相应记录。

首先建立工资变化表 Sal_log。

```
CREATE TABLE Sal_log
  (Eno NUMERIC(4) referencr teacher(eno),
   Sal NUMERIC(7,2),
   Username char(10),
   Date TIMESTAMP)
  );
CREATE TRIGGER Insert_Sal                    /* 建立了一个触发器 */
    AFTER INSERT ON Teacher                  /* 触发事件是 INSERT */
    FOR EACH ROW
     AS BEGIN
        INSERT INTO Sal_log VALUES(
            new.Eno, new.Sal, CURRENT_USER, CURRENT_TIMESTAMP);
        END;
CREATE TRIGGER Update_Sal                    /* 建立了一个触发器 */
  AFTER UPDATE ON Teacher                    /* 触发事件是 UPDATE */
  FOR EACH ROW
  AS BEGIN
      IF (new.Sal <> old.Sal) THEN INSERT INTO Sal_log VALUES(
          new.Eno, new.Sal, CURRENT_USER, CURRENT_TIMESTAMP);
      END IF;
  END;
```

5.6.2 激活触发器

触发器的执行，是由触发事件激活的，并由数据库服务器自动执行的。一个数据表上可能定义了多个触发器，比如多个 BEFORE 触发器，多个 AFTER 触发器等。同一个表上的多

个触发器激活时遵循如下的执行顺序：

（1）执行该表上的 BEFORE 触发器；

（2）激活触发器的 SQL 语句；

（3）执行该表上的 AFTER 触发器。

对于同一个表上的多个 BEFORE(AFTER)触发器，遵循“谁先创建谁先执行”的原则，即按照触发器创建的时间先后顺序执行。有些 RDBMS 是按照触发器名称的字母排序顺序执行触发器。

[例 20] 执行修改某个教师工资的 SQL 语句，激活上述定义的触发器。

```
UPDATE Teacher SET Sal = 800 WHERE Ename = '陈平';
```

执行顺序是：

- 执行触发器 Insert_Or_Update_Sal；
- 执行 SQL 语句“UPDATE Teacher SET Sal = 800 WHERE Ename = '陈平';”；
- 执行触发器 Insert_Sal；
- 执行触发器 Update_Sal。

5.6.3 删除触发器

删除触发器的 SQL 语法如下：

```
DROP TRIGGER <触发器名> ON <表名>;
```

触发器必须是一个已经创建的触发器，并且只能由具有相应权限的用户删除。

[例 21] 删除教师表 Teacher 上的触发器 Insert_Sal。

```
DROP TRIGGER Insert_Sal ON Teacher;
```

5.7 小 结

数据库的完整性是为了保证数据库中存储的数据是正确的。所谓正确的是指符合现实世界语义的。本章讲解了 RDBMS 完整性实现的机制，包括完整性约束定义机制、完整性检查机制和违背完整性约束条件时 RDBMS 应采取的动作等。

在关系系统中，最重要的完整性约束是实体完整性和参照完整性，其他完整性约束条件则可以归入用户定义的完整性。

这些数据库完整性的定义一般由 SQL 的 DDL 语句来实现。它们作为数据库模式的一部分存入数据字典中，在数据库数据修改时 RDBMS 的完整性检查机制就按照数据字典中定义的这些约束进行检查。

完整性机制的实施会影响系统性能。因此，许多数据库管理系统对完整性机制的支持比对安全性的支持要晚得多也弱得多。随着硬件性能的提高，数据库技术的发展，目前的

RDBMS 都提供了定义和检查实体完整性、参照完整性和用户定义的完整性的功能。

对于违反完整性的操作一般的处理是采用默认方式，即拒绝执行。对于违反参照完整性的操作，我们讲解了不同的处理策略。用户要根据应用语义来定义合适的处理策略，以保证数据库的正确性。

实现数据库完整性的一个重要方法是触发器。触发是定义在关系表上的由事件驱动的特殊过程。它的功能非常强，不仅可以用于数据库完整性检查，也可以用来实现数据库系统的其他功能，包括数据库安全性，以及更加广泛的应用系统的一些业务流程和控制流程，基于规则的数据和业务控制功能。

习 题

1. 什么是数据库的完整性？
2. 数据库的完整性概念与数据库的安全性概念有什么区别和联系？
3. 什么是数据库的完整性约束条件？
4. RDBMS 的完整性控制机制应具有哪些功能？
5. RDBMS 在实现参照完整性时需要考虑哪些方面？
6. 假设有下面两个关系模式：

 职工(职工号，姓名，年龄，职务，工资，部门号)，其中职工号为主码；

 部门(部门号，名称，经理名，电话)，其中部门号为主码；

用 SQL 语言定义这两个关系模式，要求在模式中完成以下完整性约束条件的定义：

(1) 定义每个模式的主码；(2) 定义参照完整性；(3) 定义职工年龄不得超过 60 岁。

7. 在关系系统中，当操作违反实体完整性、参照完整性和用户定义的完整性约束条件时，一般是如何分别进行处理的？

8. 上机实验中实现[例 18] BEFORE 行级触发器的定义，并检测触发器是否执行。

实验 4 数据控制(完整性部分)

一、实验目的

熟悉通过 SQL 对数据进行完整性控制。完成作业的上机练习。

二、实验平台

实验 1 中安装的 RDBMS 及其交互查询工具 ISQL。

三、实验内容和要求

使用 SQL 对数据进行完整性控制(三类完整性、CHECK 短语、CONSTRAIN 子句、触发器)。用实验证实，当操作违反了完整性约束条件时，系统是如何处理的。根据以下要求认真填写实验报告，记录所有的实验用例。

四、实验报告

《数据库系统概论》实验报告				
题目：	姓名		日期	
实验内容完成情况：				
出现的问题：				
解决方案(列出遇到的问题和解决办法,列出没有解决的问题)：				

本章参考文献

1 Hammer M, McLeod D. Semantic Integrity in a Relational Data Base System. in Proc. of VLDB, 1975
(本文讨论数据库管理系统中语义完整性的概念。)

2 Schmid J, Swenson J. On the Semantics of the Relational Model. in SIGMOD, 1975

3 Codd E F. Extending the Database Relational Model to Capture More Meaning. TODS, 4:4, 1979
(本文扩展了关系模型的语义表达能力。)

4 Stonebraker M, Wong E. Access Control in a Relational Database Management System by Query Modification. Proceedings of the ACM Annual Conference, 1974
(本文讨论 INGRES 数据库管理系统的完整性约束机制。)

5 Stonebraker M R. implementation of Integrity Constrains and Views by Query Modifi-cation. in Proceedings of SIGMOD, 1975
(本文讨论 INGRES 数据库管理系统的完整性约束机制。)

6 Chamberlin D, et al. SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control. IBM Journal of Research and Development, 20:6, 1976
(本文提出了 SYSTEM R 中 SQL 的 ASSERT(完整性断言)和 TRIGGER(触发器)语句。)

7 Hammer M, Sarin S. Efficient Monitoring of Database Assertions. in SIGMOD, 1978
(本文介绍了完整性断言的有效验证方法。)

8 Bernstein P, Blaustein B, Clarke E. Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data. in Proceedings of VLDB, 1980

9 Badal D, Popek G. Cost and Performance Analysis of Semantic Integrity Validation Me- thods. in SIGMOD, 1979

10 Hsu A, Imielinsky T. Integrity Checking for Multiple Updates. in SIGMOD, 1985

11 Database Language SQL. ISO/IEC 9075:1992 and ANSI X3. 135-1992, 1992

12 Database Language SQL—Part 2: Foundation (SQL/Foundation) Part 2. ANSI/ISO/IEC International Standard (IS), 1999

13 Peter Gulutzan, Trudy Pelzer 著,齐舒创作室译. SQL-99 Complete, Really, 1999

第二篇

设计与应用开发篇

本篇讲解在开发应用系统中如何在已经选定的数据库管理系统基础上设计数据库，如何基于数据库系统编程。

本篇包括3章。

第六章关系理论，本章详细讲解关系规范化理论，她既是关系数据库的重要理论基础也是数据库设计的有力工具。规范化理论为数据库设计提供了理论的指南和工具。

第七章数据库设计，这一章讨论数据库设计的方法和步骤，详细介绍了数据库设计各个阶段的目标、方法和应注意的事项。

第八章数据库编程，应用系统中可以采用多种方式来访问和管理数据库数据，本章中将讲解这些编程技术的概念和方法，包括嵌入式SQL、PL/SQL、ODBC编程等。

第六章 关系数据理论

这一章讨论关系数据理论。6.1 节从数据库逻辑设计中如何构造一个好的数据库模式这一问题出发，阐明了关系规范化理论研究的实际背景。6.2 节介绍规范化理论，讨论各种范式及可能存在的插入、删除等毛病，并直观地描述解决办法。6.3 节和 6.4 节进一步讨论关系数据理论。6.3 节讨论函数依赖的推理规则。6.4 节给出模式等价的不同定义及模式分解算法。6.1 和 6.2 两节内容是基本的，作为本科学生的要求，6.3 和 6.4 两节则可作为研究生的学习内容。

6.1 问题的提出

前面已经讨论了数据库系统的一般概念，介绍了关系数据库的基本概念、关系模型的 3 个部分以及关系数据库的标准语言 SQL。但是还有一个很基本的问题尚未涉及，针对一个具体问题，应该如何构造一个适合于它的数据模式，即应该构造几个关系模式，每个关系由哪些属性组成等。这是数据库设计的问题，确切地讲是关系数据库逻辑设计问题。

实际上设计任何一种数据库应用系统，不论是层次的、网状的还是关系的，都会遇到如何构造合适的数据模式即逻辑结构的问题。由于关系模型有严格的数学理论基础，并且可以向别的数据模型转换，因此，人们就以关系模型为背景来讨论这个问题，形成了数据库逻辑设计的一个有力工具——关系数据库的规范化理论。规范化理论虽然是以关系模型为背景，但是它对于一般的数据库逻辑设计同样具有理论上的意义。

下面首先回顾一下关系模型的形式化定义。

在第二章中已经讲过，一个关系模式应当是一个五元组。

$$R(U,D,\mathrm{DOM},F)$$

这里：

(1) 关系名 R，它是符号化的元组语义；

(2) 一组属性 U；

(3) 属性组 U 中属性所来自的域 D；

(4) 属性到域的映射 DOM;

(5) 属性组 U 上的一组数据依赖 F。

由于(3),(4)对模式设计关系不大,因此在本章中把关系模式看作是一个三元组:

$$R<U,F>$$

当且仅当 U 上的一个关系 r 满足 F 时,r 称为关系模式 $R<U,F>$ 的一个关系。

关系,作为一张二维表,对它有一个最起码的要求:每一个分量必须是不可分的数据项。满足了这个条件的关系模式就属于**第一范式**(1NF)。

在模式设计中,假设已知一个模式 Sϕ,它仅由单个关系模式组成,问题是要设计一个模式 SD,它与 Sϕ"等价"。但在某些指定的方面"更好"一些。这里通过一个例子来说明一个"不好"的模式会有些什么毛病,分析它们产生的原因,从中找出设计一个"好"的关系模式的办法。

在举例之前,先非形式地讨论一下数据依赖的概念。

数据依赖是一个关系内部属性与属性之间的一种约束关系。这种约束关系是通过属性间值的相等与否体现出来的数据间相关联系。它是现实世界属性间相互联系的抽象,是数据内在的性质,是语义的体现。

人们已经提出了许多种类型的数据依赖,其中最重要的是**函数依赖**(Functional Dependency,FD)和**多值依赖**(Multivalued Dependency,MVD)。

函数依赖极为普遍地存在于现实生活中。比如描述一个学生的关系,可以有学号(Sno)、姓名(Sname)、系名(Sdept)等几个属性。由于一个学号只对应一个学生,一个学生只在一个系学习。因而当"学号"值确定之后,学生的姓名及所在系的值也就被唯一地确定了。属性间的这种依赖关系类似于数学中的函数 $y=f(x)$,自变量 x 确定之后,相应的函数值 y 也就唯一地确定了。

类似的有 Sname = f(Sno),Sdept = f(Sno),即 Sno 函数决定 Sname,Sno 函数决定 Sdept,或者说 Sname 和 Sdept 函数依赖于 Sno,记作 Sno→Sname,Sno→Sdept。

[例 1] 现在建立一个描述学校教务的数据库,该数据库涉及的对象包括学生的学号(Sno)、所在系(Sdept)、系主任姓名(Mname)、课程号(Cno)和成绩(Grade)。假设用一个单一的关系模式 Student 来表示,则该关系模式的属性集合为:

$$U=\{\text{Sno, Sdept, Mname, Cno, Grade}\}$$

现实世界的已知事实(语义)告诉我们:

① 一个系有若干学生,但一个学生只属于一个系;

② 一个系只有一名(正职)负责人;

③ 一个学生可以选修多门课程,每门课程有若干学生选修;

④ 每个学生学习每一门课程有一个成绩。

于是得到属性组 U 上的一组函数依赖 F(如图 6.1 所示)。

$F = \{Sno \rightarrow Sdept,\ Sdept \rightarrow Mname,\ (Sno,\ Cno) \rightarrow Grade\}$

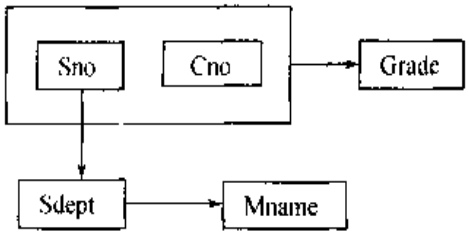


图 6.1 Student 上的一组函数依赖

如果只考虑函数依赖这一种数据依赖，我们就得到了一个描述学生的关系模式：Student $<U, F>$。表 6.1 是某一时刻关系模式 Student 的一个实例，即数据表。

表 6.1 Student 表

Sno	Sdept	Mname	Cno	Grade
S1	计算机系	张明	C1	95
S2	计算机系	张明	C1	90
S3	计算机系	张明	C1	88
S4	计算机系	张明	C1	70
S5	计算机系	张明	C1	78
⋮	⋮	⋮	⋮	⋮

但是，这个关系模式存在以下问题：

1. 数据冗余太大

比如，每一个系的系主任姓名重复出现，重复次数与该系所有学生的所有课程成绩出现次数相同，如表 6.1 所示。这将浪费大量的存储空间。

2. 更新异常(Update Anomalies)

由于数据冗余，当更新数据库中的数据时，系统要付出很大的代价来维护数据库的完整性，否则会面临数据不一致的危险。比如，某系更换系主任后，必须修改与该系学生有关的每一个元组。

3. 插入异常(Insertion Anomalies)

如果一个系刚成立，尚无学生，就无法把这个系及其系主任的信息存入数据库。

4. 删除异常(Deletion Anomalies)

如果某个系的学生全部毕业了，在删除该系学生信息的同时，把这个系及其系主任的信息也丢掉了。

鉴于存在以上种种问题，我们可以得出这样的结论：Student 关系模式不是一个好的模式。一个“好”的模式应当不会发生插入异常、删除异常、更新异常，数据冗余应尽可能少。

为什么会发生这些问题呢?

这是因为这个模式中的函数依赖存在某些不好的性质。这正是本章要讨论的问题。假如把这个单一的模式改造一下,分成3个关系模式:

S(Sno,Sdept,Sno→Sdept);
SC(Sno,Cno,Grade,(Sno,Cno)→Grade);
DEPT(Sdept, Mname, Sdept→Mname)

这3个模式都不会发生插入异常、删除异常的毛病,数据的冗余也得到了控制。

一个模式的数据依赖会有哪些不好的性质,如何改造一个不好的模式,这就是下一节规范化理论讨论的内容。

6.2 规 范 化

本节将讨论下述内容:首先讨论一个关系属性间不同的依赖情况,讨论如何根据属性间依赖情况来判定关系是否具有某些不合适的性质。通常按属性间依赖情况来区分关系规范化的程度为第一范式、第二范式、第三范式和第四范式等。然后直观地描述如何将具有不合适性质的关系转换为更合适的形式。

6.1节关系模式 Student $<U, F>$ 中有 Sno→Sdept 成立。也就是说在任何时刻 Student 的关系实例(即 Student 数据表)中,不可能存在两个元组在 Sno 上的值相等,而在 Sdept 上的值不等。因此,表6.2的 Student 表是错误的。因为表中有两个元组,Sno 上都等于 S1,而 Sdept 上一个为计算机系,一个为自动化系。

表6.2 一个错误的 Student 表

Sno	Sdept	Mname	Cno	Grade
S1	计算机系	张明	C1	95
S1	自动化系	张明	C1	90
S3	计算机系	张明	C1	88
S4	计算机系	张明	C1	70
S5	计算机系	张明	C1	78
⋮	⋮	⋮	⋮	⋮

下面给出函数依赖的一般定义。

6.2.1 函数依赖

定义6.1 设 $R(U)$ 是属性集 U 上的关系模式。X,Y 是 U 的子集。若对于 $R(U)$ 的任意一个可能的关系 r,r 中不可能存在两个元组在 X 上的属性值相等,而在 Y 上的属性值不

等,则称 X **函数确定** Y 或 Y **函数依赖于** X,记作 $X\rightarrow Y$。

函数依赖和别的数据依赖一样是语义范畴的概念。只能根据语义来确定一个函数依赖。例如 姓名→年龄 这个函数依赖只有在该部门没有同名人的条件下成立。如果允许有同名人,则年龄就不再函数依赖于姓名了。

设计者也可以对现实世界作强制的规定。例如规定不允许同名人出现,因而使 姓名→年龄 函数依赖成立。这样当插入某个元组时这个元组上的属性值必须满足规定的函数依赖,若发现有同名人存在,则拒绝插入该元组。

注意,函数依赖不是指关系模式 R 的某个或某些关系满足的约束条件,而是指 R 的一切关系均要满足的约束条件。

下面介绍一些术语和记号。

- $X\rightarrow Y$,但 $Y\nsubseteq X$ 则称 $X\rightarrow Y$ 是**非平凡的函数依赖**。
- $X\rightarrow Y$,但 $Y\subseteq X$ 则称 $X\rightarrow Y$ 是**平凡的函数依赖**。对于任一关系模式,平凡函数依赖都是必然成立的,它不反映新的语义。若不特别声明,总是讨论非平凡的函数依赖。
- 若 $X\rightarrow Y$,则 X 称为这个函数依赖的决定属性组,也称为决定因素(Determinant)。
- 若 $X\rightarrow Y$,$Y\rightarrow X$,则记作 $X\longleftrightarrow Y$。
- 若 Y 不函数依赖于 X,则记作 $X\nrightarrow Y$。

定义 6.2 在 $R(U)$ 中,如果 $X\rightarrow Y$,并且对于 X 的任何一个真子集 X',都有 $X'\nrightarrow Y$,则称 Y 对 X **完全函数依赖**,记作

$$X\xrightarrow{F}Y$$

若 $X\rightarrow Y$,但 Y 不完全函数依赖于 X,则称 Y 对 X **部分函数依赖**,记作:

$$X\xrightarrow{P}Y$$

[**例 1**]中(Sno, Cno)$\xrightarrow{F}$Grade 是完全函数依赖,(Sno, Cno)$\xrightarrow{P}$Sdept 是部分函数依赖,因为 Sno→Sdept 成立,而 Sno 是(Sno, Cno)的真子集。

定义 6.3 在 $R(U)$ 中,如果 $X\rightarrow Y$,$(Y\nsubseteq X)$,$Y\nrightarrow X$,$Y\rightarrow Z$,则称 Z 对 X **传递函数依赖**。记为:$X\xrightarrow{传递}Z$。

[**例 1**]中有 Sno→Sdept, Sdept→Mname 成立,所以 Sno$\xrightarrow{传递}$Mname。

加上条件 $Y\nrightarrow X$,是因为如果 $Y\rightarrow X$,则 $X\longleftrightarrow Y$,实际上是 $X\xrightarrow{直接}Z$,是**直接函数依赖**而不是传递函数依赖。

6.2.2 码

码是关系模式中一个重要概念。在第二章中已给出了有关码的若干定义。这里用函数依赖的概念来定义码。

定义 6.4 设 K 为 $\boldsymbol{R}<U,F>$ 中的属性或属性组合，若 $K \xrightarrow{F} U$ 则 K 为 $\boldsymbol{R}$ 的**候选码**(Candidate key)。若候选码多于一个，则选定其中的一个为**主码**(Primary key)。

包含在任何一个候选码中的属性，称为**主属性**(Prime attribute)。不包含在任何码中的属性称为**非主属性**(Nonprime attribute)或**非码属性**(Non-key attribute)。最简单的情况，单个属性是码。最极端的情况，整个属性组是码，称为**全码**(All-key)。

[**例 2**] 关系模式 S(Sno,Sdept,Sage)中单个属性 Sno 是码，用下横线显示出来。SC(Sno,Cno,Grade)中属性组合(Sno,Cno)是码。

[**例 3**] 关系模式 R(P,W,A)，属性 P 表示演奏者，W 表示作品，A 表示听众。假设一个演奏者可以演奏多个作品，某一作品可被多个演奏者演奏。听众也可以欣赏不同演奏者的不同作品，这个关系模式的码为(P,W,A)，即 All-key。

定义 6.5 关系模式 $\boldsymbol{R}$ 中属性或属性组 X 并非 $\boldsymbol{R}$ 的码，但 X 是另一个关系模式的码，则称 X 是 $\boldsymbol{R}$ 的**外部码**(Foreign key)，也称**外码**。

如在 SC(Sno,Cno,Grade)中，Sno 不是码，但 Sno 是关系模式 S(Sno,Sdept,Sage)的码，则 Sno 是关系模式 SC 的外部码。

主码与外部码提供了一个表示关系间联系的手段。如[**例 2**]中关系模式 S 与 SC 的联系就是通过 Sno 来体现的。

6.2.3 范式

关系数据库中的关系是要满足一定要求的，满足不同程度要求的为不同范式。满足最低要求的叫第一范式，简称 1NF。在第一范式中满足进一步要求的为第二范式，其余以此类推。

从范式来讲，主要是 E. F. Codd 做的工作，1971～1972 年他系统地提出了 1NF、2NF、3NF 的概念，讨论了规范化的问题。1974 年，Codd 和 Boyce 又共同提出了一个新范式，即 BCNF。1976 年 Fagin 又提出了 4NF。后来又有人提出了 5NF。

本来，所谓"第几范式"，是表示关系的某一种级别。所以经常称某一关系模式 $\boldsymbol{R}$ 为第几范式。现在把范式这个概念理解成符合某一种级别的关系模式的集合，则 $\boldsymbol{R}$ 为第几范式就可以写成 $\boldsymbol{R} \in x$ NF。

对于各种范式之间的联系有

$$5NF \subset 4NF \subset BCNF \subset 3NF \subset 2NF \subset 1NF$$

成立，如图 6.2 所示。

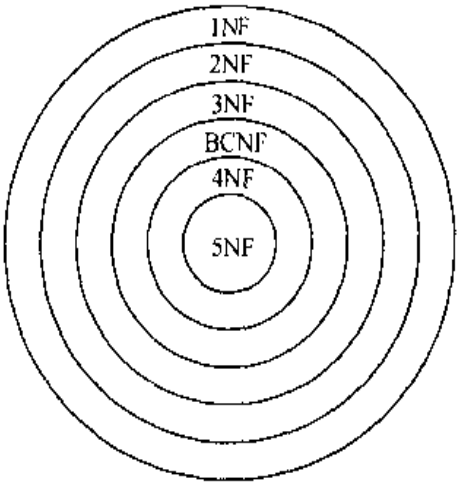


图 6.2 各种范式之间的联系

一个低一级范式的关系模式，通过模式分解可以转换为若干个高一级范式的关系模式的集合，这种过程就叫规范化。

6.2.4 2NF

定义 6.6 若 $R \in 1NF$，且每一个非主属性完全函数依赖于码，则 $R \in 2NF$。

下面举一个不是 2NF 的例子。

[例 4] 关系模式 S-L-C(Sno,Sdept,Sloc,Cno,Grade)

某中 SLOC 为学生的住处，并且每个系的学生住在同一个地方。S-L-C 的码为(Sno,Cno)。函数依赖有：

$$(Sno,Cno)\xrightarrow{F}Grade$$

$$Sno \rightarrow Sdept,(Sno,Cno)\xrightarrow{P}Sdept$$

$$Sno \rightarrow Sloc,(Sno,Cno)\xrightarrow{P}Sloc,$$

Sdept→Sloc(因为每个系的学生只住一个地方)，如图 6.3 所示。

图中用虚线表示部分函数依赖。另外，SDEPT 还函数决定 Sloc，这一点在讨论第二范式时暂不考虑。可以看到非主属性 Sdept、Sloc 并不完全函数依赖于码。因此 S-L-C(Sno,Sdept,Sloc,Cno,Grade)不符合 2NF 定义，即 S-L-C $\notin$ 2NF。

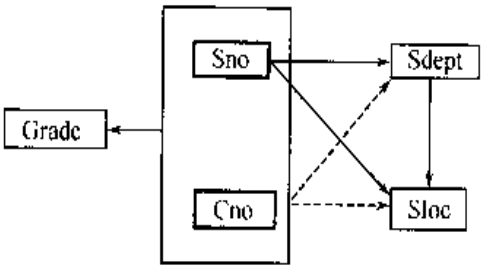


图 6.3 函数依赖示例

一个关系模式 R 不属于 2NF，就会产生以下几个问题：

1. 插入异常。假若要插入一个学生 Sno = S7，Sdept = PHY，Sloc = BLD2，但该生还未选课，即这个学生无 Cno，这样的元组就插不进 S-L-C 中。因为插入元组时必须给定码值，而这时码值的一部分为空，因而学生的固有信息无法插入。

2. 删除异常。假定某个学生只选一门课，如 S4 就选了一门课 C3。现在 C3 这门课他也不选了，那么 C3 这个数据项就要删除。而 C3 是主属性，删除了 C3，整个元组就必须跟着删除，使得 S4 的其他信息也被删除了，从而造成删除异常，即不应删除的信息也删除了。

3. 修改复杂。某个学生从数学系(MA)转到计算机科学系(CS)，这本来只需修改此学生元组中的 Sdept 分量。但因为关系模式 S-L-C 中还含有系的住处 Sloc 属性，学生转系将同时改变住处，因而还必须修改元组中的 Sloc 分量。另外，如果这个学生选修了 k 门课，Sdept，Sloc 重复存储了 k 次，不仅存储冗余度大，而且必须无遗漏地修改 k 个元组中全部 Sdept，Sloc 信息，造成修改的复杂化。

分析上面的例子，可以发现问题在于有两种非主属性。一种如 Grade，它对码是完全函数依赖。另一种如 Sdept、Sloc 对码不是完全函数依赖。解决的办法是用投影分解把关系模式 S-L-C 分解为两个关系模式。

SC(Sno,Cno,Grade)

S-L(Sno,Sdept,Sloc)

关系模式 SC 与 S-L 中属性间的函数依赖可以用图 6.4、图 6.5 表示如下：

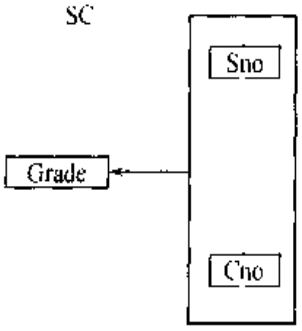


图 6.4 SC 中的函数依赖

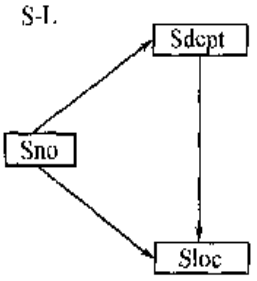


图 6.5 S-L 中的函数依赖

关系模式 SC 的码为(Sno,Cno)，关系模式 S-L 的码为 Sno，这样就使得非主属性对码都是完全函数依赖了。

6.2.5 3NF

定义 6.7 关系模式 $R<U,F>$ 中若不存在这样的码 X，属性组 Y 及非主属性 $Z(Z\subseteq Y)$ 使得 $X\rightarrow Y$，$Y\rightarrow Z$ 成立，$Y\not\rightarrow X$，则称 $R<U,F>\in 3NF$。

由**定义 6.7** 可以证明，若 $R\in 3NF$，则每一个非主属性既不部分依赖于码也不传递依赖于码。

在图 6.4 中关系模式 SC 没有传递依赖，而图 6.5 中关系模式 S-L 存在非主属性对码的传递依赖。在 S-L 中，由 Sno→Sdept，(Sdept↛Sno)，Sdept→Sloc，可得 $\text{Sno}\xrightarrow{\text{传递}}\text{Sloc}$。因此 $\text{SC}\in 3NF$，而 $\text{S-L}\notin 3NF$。

一个关系模式 R 若不是 3NF，就会产生与 6.2.4 节中 2NF 相类似的问题。读者可以类比 2NF 的反例加以说明。

解决的办法同样是将 S-L 分解为：

S-D(Sno,Sdept)

D-L(Sdept,Sloc)

分解后的关系模式 S-D 与 D-L 中不再存在传递依赖。

6.2.6 BCNF

BCNF(Boyce Codd Normal Form)是由 Boyce 与 Codd 提出的，比上述的 3NF 又进了一步，通常认为 BCNF 是修正的第三范式，有时也称为扩充的第三范式。

定义 6.8 关系模式 $R<U,F>\in 1NF$。若 $X\rightarrow Y$ 且 $Y\subseteq X$ 时 X 必含有码，则 $R<U,F>\in BCNF$。

也就是说，关系模式 $R<U,F>$ 中，若每一个决定因素都包含码，则 $R<U,F>\in$ BCNF。

由 BCNF 的定义可以得到结论，一个满足 BCNF 的关系模式有：

- 所有非主属性对每一个码都是完全函数依赖；
- 所有的主属性对每一个不包含它的码，也是完全函数依赖；
- 没有任何属性完全函数依赖于非码的任何一组属性。

由于 $R\in$ BCNF，按定义排除了任何属性对码的传递依赖与部分依赖，所以 $R\in$ 3NF。严格的证明留给读者完成。但是若 $R\in$ 3NF，则 R 未必属于 BCNF。

下面用几个例子说明属于 3NF 的关系模式有的属于 BCNF，但有的不属于 BCNF。

[例 5] 考察关系模式 C(Cno,Cname,Pcno)，它只有一个码 Cno，这里没有任何属性对 Cno 部分依赖或传递依赖，所以 C∈3NF。同时 C 中 Cno 是唯一的决定因素，所以 C∈BCNF。对于关系模式 SC(Sno,Cno,Grade)可作同样分析。

[例 6] 关系模式 S(Sno,Sname,Sdept,Sage)，假定 Sname 也具有唯一性，那么 S 就有两个码，这两个码都由单个属性组成，彼此不相交。其他属性不存在对码的传递依赖与部分依赖，所以 S∈3NF。同时 S 中除 Sno,Sname 外没有其他决定因素，所以 S 也属于 BCNF。

以下再举几个例子

[例 7] 关系模式 SJP(S,J,P)中，S 是学生，J 表示课程，P 表示名次。每一个学生选修每门课程的成绩有一定的名次，每门课程中每一名次只有一个学生(即没有并列名次)。由语义可得到下面的函数依赖：

$$(S,J)\rightarrow P;(J,P)\rightarrow S$$

所以(S,J)与(J,P)都可以作为候选码。这两个码各由两个属性组成，而且它们是相交的。这个关系模式中显然没有属性对码传递依赖或部分依赖。所以 SJP∈3NF，而且除(S,J)与(J,P)以外没有其他决定因素，所以 SJP∈BCNF。

[例 8] 关系模式 STJ(S,T,J)中，S 表示学生，T 表示教师，J 表示课程。每一教师只教一门课。每门课有若干教师，某一学生选定某门课，就对应一个固定的教师。由语义可得到如下的函数依赖。

(S,J)→T;(S,T)→J;T→J，用图 6.6 表示。

这里(S,J)、(S,T)都是候选码。

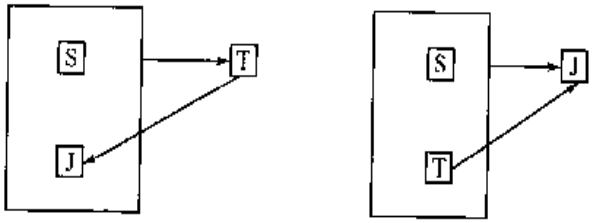


图 6.6 STJ 中的函数依赖

STJ 是 3NF,因为没有任何非主属性对码传递依赖或部分依赖。但 STJ 不是 BCNF 关系,因为 T 是决定因素,而 T 不包含码。

对于不是 BCNF 的关系模式,仍然存在不合适的地方。读者可自己举例指出 STJ 的不合适之处。非 BCNF 的关系模式也可以通过分解成为 BCNF。例如 STJ 可分解为 ST(S,T)与 TJ(T,J),它们都是 BCNF。

3NF 和 BCNF 是在函数依赖的条件下对模式分解所能达到的分离程度的测度。一个模式中的关系模式如果都属于 BCNF,那么在函数依赖范畴内,它已实现了彻底的分离,已消除了插入和删除的异常。3NF 的"不彻底"性表现在可能存在主属性对码的部分依赖和传递依赖。

6.2.7 多值依赖

以上完全是在函数依赖的范畴内讨论问题。属于 BCNF 的关系模式是否就很完美了呢?下面来看一个例子。

[例 9] 学校中某一门课程由多个教员讲授,他们使用相同的一套参考书。每个教员可以讲授多门课程,每种参考书可以供多门课程使用。可以用一个非规范化的关系来表示教员 T、课程 C 和参考书 B 之间的关系(如表 6.3 所示)。

表 6.3 非规范化关系示例

课程 C	教员 T	参考书 B
物理	{李勇 王军}	{普通物理学 光学原理 物理习题集}
数学	{李勇 张平}	{数学分析 微分方程 高等代数}
计算数学	{张平 周峰}	{数学分析 … …}
⋮	⋮	⋮

把这张表变成一张规范化的二维表,如表 6.4 所示。

关系模型 Teaching(C,T,B)的码是(C,T,B),即 All-Key。因而 Teaching ∈ BCNF。但是当某一课程(如物理)增加一名讲课教员(如周英)时,必须插入多个(这里是 3 个)元组:(物理,周英,普通物理学);(物理,周英,光学原理);(物理,周英,物理习题集)。

同样,某一门课(如数学)要去掉一本参考书(如微分方程),则必须删除多个(这里是两个)元组:(数学,李勇,微分方程);(数学,张平,微分方程)。

对数据的增删改很不方便，数据的冗余也十分明显。仔细考察这类关系模式，发现它具有一种称之为多值依赖(MVD)的数据依赖。

定义 6.9 设 $R(U)$ 是属性集 U 上的一个关系模式。X,Y,Z 是 U 的子集，并且 $Z=U-X-Y$。关系模式 $R(U)$ 中多值依赖 $X\rightarrow\rightarrow Y$ 成立，当且仅当对 $R(U)$ 的任一关系 r，给定的一对 (x,z) 值，有一组 Y 的值，这组值仅仅决定于 x 值而与 z 值无关。

表 6.4 Teaching

课 程 C	教员 T	参考书 B
物 理	李 勇	普通物理学
物 理	李 勇	光学原理
物 理	李 勇	物理习题集
物 理	王 军	普通物理学
物 理	王 军	光学原理
物 理	王 军	物理习题集
数 学	李 勇	数学分析
数 学	李 勇	微分方程
数 学	李 勇	高等代数
数 学	张 平	数学分析
数 学	张 平	微分方程
数 学	张 平	高等代数
⋮	⋮	⋮

例如，在关系模式 Teaching 中，对于一个(物理，光学原理)有一组 T 值{李勇，王军}，这组值仅仅决定于课程 C 上的值(物理)。也就是说对于另一个(物理，普通物理学)它对应的一组 T 值仍是{李勇，王军}，尽管这时参考书 B 的值已经改变了。因此 T 多值依赖于 C，即 $C\rightarrow\rightarrow T$。

对于多值依赖的另一个等价的形式化的定义是：

在 $R(U)$ 的任一关系 r 中，如果存在元组 t,s 使得 $t[X]=s[X]$，那么就必然存在元组 $w,v\in r$，(w,v 可以与 s,t 相同)，使得 $w[X]=v[X]=t[X]$，而 $w[Y]=t[Y]$，$w[Z]=s[Z]$，$v[Y]=s[Y]$，$v[Z]=t[Z]$(即交换 s,t 元组的 Y 值所得的两个新元组必在 r 中)，则 Y 多值依赖于 X，记为 $X\rightarrow\rightarrow Y$。这里，$X,Y$ 是 U 的子集，$Z=U-X-Y$。

若 $X\rightarrow\rightarrow Y$，而 $Z=\phi$ 即 Z 为空，则称 $X\rightarrow\rightarrow Y$ 为**平凡的多值依赖**。

下面再举一个具有多值依赖的关系模式的例子。

[例 10] 关系模式 WSC(W,S,C)中，W 表示仓库，S 表示保管员，C 表示商品。假设每个仓库有若干个保管员，有若干种商品。每个保管员保管所在的仓库的所有商品，每种商品被所有保管员保管。列出关系如下：

W	S	C
W1	S1	C1
W1	S1	C2
W1	S1	C3
W1	S2	C1
W1	S2	C2
W1	S2	C3
W2	S3	C4
W2	S3	C5
W2	S4	C4
W2	S4	C5

按照语义对于 W 的每一个值 W_i，S 有一个完整的集合与之对应而不问 C 取何值。所以 W→→S。

如果用图 6.7 来表示这种对应，则对应 W 的某一个值 W_i 的全部 S 值记作 $|S|_{W_i}$（表示此仓库工作的全部保管员），全部 C 值记作 $|C|_{W_i}$（表示在此仓库中存放的所有商品）。应当有 $|S|_{W_i}$ 中的每一个值和 $|C|_{W_i}$ 中的每一个 C 值对应。于是 $|S|_{W_i}$ 与 $|C|_{W_i}$ 之间正好形成一个完全二分图，因而 W→→S。

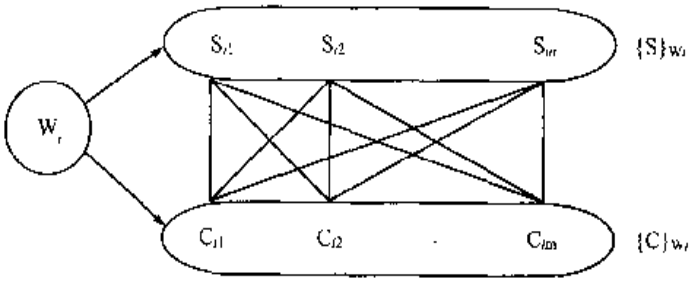


图 6.7 W→→S 且 W→→C

由于 C 与 S 的完全对称性，必然有 W→→C 成立。

多值依赖具有以下性质：

（1）多值依赖具有对称性。即若 $X \to\to Y$，则 $X \to\to Z$，其中 $Z = U - X - Y$。

从[**例 10**]容易看出，因为每个保管员保管所有商品，同时每种商品被所有保管员保管，显然若 W→→S，必然有 W→→C。

（2）多值依赖的传递性。即若 $X \to\to Y$，$Y \to\to Z$，则 $X \to\to Z - Y$。

（3）函数依赖可以看作是多值依赖的特殊情况。即若 $X \to Y$，则 $X \to\to Y$。这是因为当 $X \to Y$ 时，对 X 的每一个值 x，Y 有一个确定的值 y 与之对应，所以 $X \to\to Y$。

（4）若 $X \to\to Y$，$X \to\to Z$，则 $X \to\to YZ$。

（5）若 $X \to\to Y$，$X \to\to Z$，则 $X \to\to Y \cap Z$。

(6) 若 $X\rightarrow\rightarrow Y, X\rightarrow\rightarrow Z$,则 $X\rightarrow\rightarrow Y-Z, X\rightarrow\rightarrow Z-Y$。

多值依赖与函数依赖相比,具有下面两个基本的区别:

(1) 多值依赖的有效性与属性集的范围有关。

若 $X\rightarrow\rightarrow Y$ 在 U 上成立则在 $W(XY\subseteq W\subseteq U)$ 上一定成立;反之则不然,即 $X\rightarrow\rightarrow Y$ 在 $W(W\subset U)$ 上成立,在 U 上并不一定成立。这是因为多值依赖的定义中不仅涉及属性组 X 和 Y,而且涉及 U 中其余属性 Z。

一般地,在 $\boldsymbol{R}(U)$ 上若有 $X\rightarrow\rightarrow Y$ 在 $W(W\subset U)$ 上成立,则称 $X\rightarrow\rightarrow Y$ 为 $\boldsymbol{R}(U)$ 的嵌入型多值依赖。

但是在关系模式 $\boldsymbol{R}(U)$ 中函数依赖 $X\rightarrow Y$ 的有效性仅决定于 X, Y 这两个属性集的值。只要在 $\boldsymbol{R}(U)$ 的任何一个关系 r 中,元组在 X 和 Y 上的值满足**定义 6.1**,则函数依赖 $X\rightarrow Y$ 在任何属性集 $W(XY\subseteq W\subseteq U)$ 上成立。

(2) 若函数依赖 $X\rightarrow Y$ 在 $\boldsymbol{R}(U)$ 上成立,则对于任何 $Y'\subset Y$ 均有 $X\rightarrow Y'$ 成立。而多值依赖 $X\rightarrow\rightarrow Y$ 若在 $\boldsymbol{R}(U)$ 上成立,却不能断言对于任何 $Y'\subset Y$ 有 $X\rightarrow\rightarrow Y'$ 成立。

6.2.8 4NF

定义 6.10 关系模式 $\boldsymbol{R}<U,F>\in$ 1NF,如果对于 $\boldsymbol{R}$ 的每个非平凡多值依赖 $X\rightarrow\rightarrow Y(Y\subseteq X)$,$X$ 都含有码,则称 $\boldsymbol{R}<U,F>\in$ 4NF。

4NF 就是限制关系模式的属性之间不允许有非平凡且非函数依赖的多值依赖。因为根据定义,对于每一个非平凡的多值依赖 $X\rightarrow\rightarrow Y$,$X$ 都含有候选码,于是就有 $X\rightarrow Y$,所以 4NF 所允许的非平凡的多值依赖实际上是函数依赖。

显然,如果一个关系模式是 4NF,则必为 BCNF。

在前面讨论的关系模式 WSC 中,$W\rightarrow\rightarrow S$,$W\rightarrow\rightarrow C$,它们都是非平凡的多值依赖。而 W 不是码,关系模式 WSC 的码是(W,S,C),即 All-Key。因此 WSC $\notin$ 4NF。

一个关系模式如果已达到了 BCNF 但不是 4NF,这样的关系模式仍然具有不好的性质。以 WSC 为例,WSC $\notin$ 4NF,但是 WSC $\in$ BCNF。对于 WSC 的某个关系,若某一仓库 W_i 有 n 个保管员,存放 m 件物品,则关系中分量为 W_i 的元组数目一定有 $m\times n$ 个。每个保管员重复存储 m 次,每种物品重复存储 n 次,数据的冗余度太大,因此还应该继续规范化使关系模式 WSC 达到 4NF。

可以用投影分解的方法消去非平凡且非函数依赖的多值依赖。例如可以把 WSC 分解为 WS(W,S),WC(W,C)。在 WS 中虽然有 $W\rightarrow\rightarrow S$,但这是平凡的多值依赖。WS 中已不存在非平凡的非函数依赖的多值依赖。所以 WS $\in$ 4NF,同理 WC $\in$ 4NF。

函数依赖和多值依赖是两种最重要的数据依赖。如果只考虑函数依赖,则属于 BCNF 的关系模式规范化程度已经是最高的了。如果考虑多值依赖,则属于 4NF 的关系模式规范化程度是最高的。事实上,数据依赖中除函数依赖和多值依赖之外,还有其他数据依赖。例

如有一种连接依赖。函数依赖是多值依赖的一种特殊情况，而多值依赖实际上又是连接依赖的一种特殊情况。但连接依赖不像函数依赖和多值依赖可由语义直接导出，而是在关系的连接运算时才反映出来。存在连接依赖的关系模式仍可能遇到数据冗余及插入、修改、删除异常等问题。如果消除了属于4NF的关系模式中存在的连接依赖，则可以进一步达到5NF的关系模式。这里不再讨论连接依赖和5NF，有兴趣的读者可以参阅有关书籍。

6.2.9 规范化小结

在关系数据库中，对关系模式的基本要求是满足第一范式。这样的关系模式就是合法的、允许的。但是，人们发现有些关系模式存在插入、删除异常，修改复杂，数据冗余等毛病。人们寻求解决这些问题的方法，这就是规范化的目的。

规范化的基本思想是逐步消除数据依赖中不合适的部分，使模式中的各关系模式达到某种程度的"分离"，即"一事一地"的模式设计原则。让一个关系描述一个概念、一个实体或者实体间的一种联系。若多于一个概念就把它"分离"出去。因此所谓规范化实质上是概念的**单一化**。

人们认识这个原则是经历了一个过程的。从认识非主属性的部分函数依赖的危害开始，2NF，3NF，BCNF，4NF的提出是这个认识过程逐步深化的标志，图6.8可以概括这个过程。

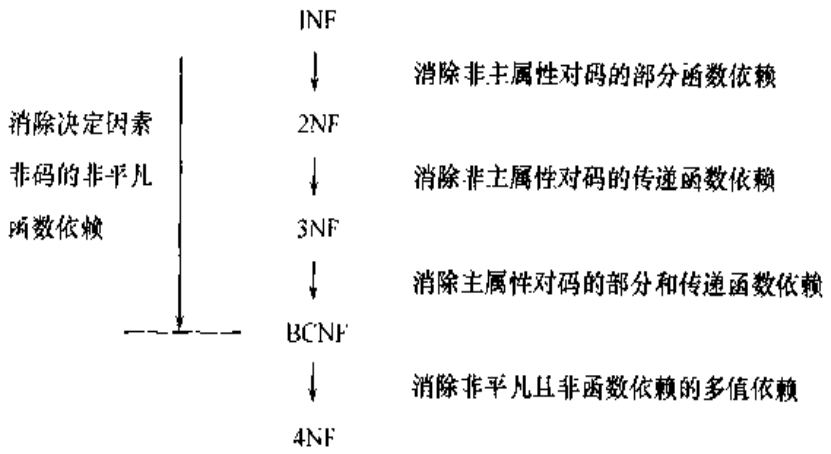


图6.8 规范化过程

关系模式的规范化过程是通过对关系模式的分解来实现的。把低一级的关系模式分解为若干个高一级的关系模式。这种分解不是唯一的。下面就将进一步讨论分解后的关系模式与原关系模式"等价"的问题以及分解的算法。

6.3 数据依赖的公理系统

数据依赖的公理系统是模式分解算法的理论基础，下面首先讨论函数依赖的一个有效

而完备的公理系统——Armstrong 公理系统。

定义 6.11 对于满足一组函数依赖 F 的关系模式 $\boldsymbol{R}<U,F>$，其任何一个关系 r，若函数依赖 $X\rightarrow Y$ 都成立（即 r 中任意两元组 t,s，若 $t[X]=s[X]$，则 $t[Y]=s[Y]$），则称 $\boldsymbol{F}$ **逻辑蕴含** $\boldsymbol{X\rightarrow Y}$。

为了求得给定关系模式的码，为了从一组函数依赖求得蕴含的函数依赖，例如已知函数依赖集 F，要问 $X\rightarrow Y$ 是否为 F 所蕴含，就需要一套推理规则，这组推理规则是 1974 年首先由 Armstrong 提出来的。

Armstrong 公理系统 设 U 为属性集总体，F 是 U 上的一组函数依赖，于是有关系模式 $\boldsymbol{R}<U,F>$。对 $\boldsymbol{R}<U,F>$ 来说有以下的推理规则：

- A1. **自反律**（Reflexivity）：若 $Y\subseteq X\subseteq U$，则 $X\rightarrow Y$ 为 F 所蕴含。
- A2. **增广律**（Augmentation）：若 $X\rightarrow Y$ 为 F 所蕴含，且 $Z\subseteq U$，则 $XZ\rightarrow YZ$[①] 为 F 所蕴含。
- A3. **传递律**（Transitivity）：若 $X\rightarrow Y$ 及 $Y\rightarrow Z$ 为 F 所蕴含，则 $X\rightarrow Z$ 为 F 所蕴含。

注意：由自反律所得到的函数依赖均是平凡的函数依赖，自反律的使用并不依赖于 F。

定理 6.1 Armstrong 推理规则是正确的。

下面从定义出发证明推理规则的正确性。

证

（1）设 $Y\subseteq X\subseteq U$。

对 $\boldsymbol{R}<U,F>$ 的任一关系 r 中的任意两个元组 t,s：

若 $t[X]=s[X]$，由于 $Y\subseteq X$，有 $t[Y]=s[Y]$，

所以 $X\rightarrow Y$ 成立，自反律得证[②]。

（2）设 $X\rightarrow Y$ 为 F 所蕴含，且 $Z\subseteq U$。

设 $\boldsymbol{R}<U,F>$ 的任一关系 r 中任意的两个元组 t,s：

若 $t[XZ]=s[XZ]$，则有 $t[X]=s[X]$ 和 $t[Z]=s[Z]$；

由 $X\rightarrow Y$，于是有 $t[Y]=s[Y]$，所以 $t[YZ]=s[YZ]$，所以 $XZ\rightarrow YZ$ 为 F 所蕴含，增广律得证。

（3）设 $X\rightarrow Y$ 及 $Y\rightarrow Z$ 为 F 所蕴含。

对 $\boldsymbol{R}<U,F>$ 的任一关系 r 中的任意两个元组 t,s：

若 $t[X]=s[X]$，由于 $X\rightarrow Y$，有 $t[Y]=s[Y]$；

再由 $Y\rightarrow Z$，有 $t[Z]=s[Z]$，所以 $X\rightarrow Z$ 为 F 所蕴含，传递律得证。

根据 A1、A2、A3 这 3 条推理规则可以得到下面三条很有用的推理规则：

- **合并规则**：由 $X\rightarrow Y,X\rightarrow Z$，有 $X\rightarrow YZ$。
- **伪传递规则**：由 $X\rightarrow Y,WY\rightarrow Z$，有 $XW\rightarrow Z$。

① 为了简单起见，用 XZ 代表 $X\cup Z$。

② $t[X]$ 表示元组 t 在属性（组）X 上的分量，等价于 $t.X$。

• **分解规则**:由 $X\rightarrow Y$ 及 $Z\subseteq Y$,有 $X\rightarrow Z$。

根据合并规则和分解规则,很容易得到这样一个重要事实:

引理 6.1 $X\rightarrow A_1A_2\cdots A_k$ 成立的充分必要条件是 $X\rightarrow A_i$ 成立($i=1,2,\cdots,k$)。

定义 6.12 在关系模式 $\boldsymbol{R}<U,F>$ 中为 F 所逻辑蕴含的函数依赖的全体叫作 F 的**闭包**,记为 F^+。

人们把自反律,传递律和增广律称为 Armstrong 公理系统。Armstrong 公理系统是有效的、完备的。Armstrong 公理的**有效性**指的是:由 F 出发根据 Armstrong 公理推导出来的每一个函数依赖一定在 F^+ 中;**完备性**指的是 F^+ 中的每一个函数依赖,必定可以由 F 出发根据 Armstrong 公理推导出来。

要证明完备性,就首先要解决如何判定一个函数依赖是否属于由 F 根据 Armstrong 公理推导出来的函数依赖的集合。当然,如果能求出这个集合,问题就解决了。但不幸的是,这是一个 NP 完全问题。比如从 $F=\{X\rightarrow A_1,\cdots,X\rightarrow A_n\}$ 出发,至少可以推导出 2^n 个不同的函数依赖。为此引入了下面的概念:

定义 6.13 设 F 为属性集 U 上的一组函数依赖,$X\subseteq U$,$X_F^+=\{A\mid X\rightarrow A$ 能由 F 根据 Armstrong 公理导出$\}$,X_F^+ 称为属性集 X **关于函数依赖集 F 的闭包**。

由**引理 6.1** 容易得出:

引理 6.2 设 F 为属性集 U 上的一组函数依赖,$X,Y\subseteq U$,$X\rightarrow Y$ 能由 F 根据 Armstrong 公理导出的充分必要条件是 $Y\subseteq X_F^+$。

于是,判定 $X\rightarrow Y$ 是否能由 F 根据 Armstrong 公理导出的问题,就转化为求出 X_F^+,判定 Y 是否为 X_F^+ 的子集的问题。这个问题由**算法 6.1** 解决了。

算法 6.1 求属性集 $X(X\subseteq U)$ 关于 U 上的函数依赖集 F 的闭包 X_F^+。

输入:X,F

输出:X_F^+

步骤:

(1) 令 $X^{(0)}=X$,$i=0$

(2) 求 B,这里 $B=\{A\mid(\exists V)(\exists W)(V\rightarrow W\in F\wedge V\subseteq X^{(i)}\wedge A\in W)\}$;

(3) $X^{(i+1)}=B\cup X^{(i)}$

(4) 判断 $X^{(i+1)}=X^{(i)}$ 吗?

(5) 若相等或 $X^{(i)}=U$ 则 $X^{(i)}$ 就是 X_F^+,算法终止。

(6) 若否,则 $i=i+1$,返回第(2)步。

[例 1] 已知关系模式 $\boldsymbol{R}<U,F>$,

其中 $U=\{A,B,C,D,E\}$;$F=\{AB\rightarrow C,B\rightarrow D,C\rightarrow E,EC\rightarrow B,AC\rightarrow B\}$。

求 $(AB)_F^+$。

解 由**算法 6.1**,设 $X^{(0)}=AB$;

计算 $X^{(1)}$；逐一的扫描 F 集合中各个函数依赖，找左部为 A，B 或 AB 的函数依赖。得到两个：$AB\to C$，$B\to D$。于是 $X^{(1)}=AB\cup CD=ABCD$。

因为 $X^{(0)}\neq X^{(1)}$，所以再找出左部为 $ABCD$ 子集的那些函数依赖，又得到 $C\to E$，$AC\to B$，于是 $X^{(2)}=X^{(1)}\cup BE=ABCDE$。

因为 $X^{(2)}$ 已等于全部属性集合，所以 $(AB)_F^+=ABCDE$。

对于**算法 6.1**，令 $a_i=|X^{(i)}|$，$\{a_i\}$ 形成一个步长大于 1 的严格递增的序列，序列的上界是 $|U|$，因此该算法最多 $|U|-|X|$ 次循环就会终止。

定理 6.2 Armstrong 公理系统是有效的、完备的。

Armstrong 公理系统的有效性可由**定理 6.1** 得到证明。这里给出完备性的证明。

证明完备性的逆否命题，即若函数依赖 $X\to Y$ 不能由 F 从 Armstrong 公理导出，那么它必然不为 F 所蕴含，它的证明分三步。

(1) 若 $V\to W$ 成立，且 $V\subseteq X_F^+$，则 $W\subseteq X_F^+$。

证 因为 $V\subseteq X_F^+$，所以有 $X\to V$ 成立；于是 $X\to W$ 成立(因为 $X\to V$，$V\to W$)，所以 $W\subseteq X_F^+$。

(2) 构造一张二维表 r，它由下列两个元组构成，可以证明 r 必是 $\boldsymbol{R}(U,F)$ 的一个关系，即 F 中的全部函数依赖在 r 上成立。

X_F^+	$U-X_F^+$
$11\cdots\cdots1$	$00\cdots\cdots0$
$11\cdots\cdots1$	$11\cdots\cdots1$

若 r 不是 $\boldsymbol{R}<U,F>$ 的关系，则必由于 F 中有某一个函数依赖 $V\to W$ 在 r 上不成立所致。由 r 的构成可知，V 必定是 X_F^+ 的子集，而 W 不是 X_F^+ 的子集，可是由第(1)步，$W\subseteq X_F^+$，矛盾。所以 r 必是 $\boldsymbol{R}<U,F>$ 的一个关系。

(3) 若 $X\to Y$ 不能由 F 从 Armstrong 公理导出，则 Y 不是 X_F^+ 的子集，因此必有 Y 的子集 Y' 满足 $Y'\subseteq U-X_F^+$，则 $X\to Y$ 在 r 中不成立，即 $X\to Y$ 必不为 $\boldsymbol{R}<U,F>$ 蕴含。

Armstrong 公理的完备性及有效性说明了“导出”与“蕴含”是两个完全等价的概念。于是 F^+ 也可以说成是由 F 出发借助 Armstrong 公理导出的函数依赖的集合。

从蕴含(或导出)的概念出发，又引出了两个函数依赖集等价和最小依赖集的概念。

定义 6.14 如果 $G^+=F^+$，就说函数依赖集 F 覆盖 G(F 是 G 的覆盖，或 G 是 F 的覆盖)，或 F 与 G 等价。

引理 6.3 $F^+=G^+$ 的充分必要条件是 $F\subseteq G^+$，和 $G\subseteq F^+$。

证 必要性显然，只证充分性。

(1) 若 $F\subseteq G^+$，则 $X_F^+\subseteq X_{G^+}^+$。

(2) 任取 $X\to Y\in F^+$ 则有 $Y\subseteq X_F^+\subseteq X_{G^+}^+$。

所以 $X\to Y\in(G^+)^+=G^+$。即 $F^+\subseteq G^+$。

(3) 同理可证 $G^+ \subseteq F^+$,所以 $F^+ = G^+$。

而要判定 $F \subseteq G^+$,只须逐一对 F 中的函数依赖 $X \rightarrow Y$,考察 Y 是否属于 X_G^+,就行了。因此**引理 6.3** 给出了判断两个函数依赖集等价的可行算法。

定义 6.15 如果函数依赖集 F 满足下列条件,则称 F 为一个极小函数依赖集。亦称为最小依赖集或最小覆盖。

(1) F 中任一函数依赖的右部仅含有一个属性。

(2) F 中不存在这样的函数依赖 $X \rightarrow A$,使得 F 与 $F - \{X \rightarrow A\}$ 等价。

(3) F 中不存在这样的函数依赖 $X \rightarrow A$,X 有真子集 Z 使得 $F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ 与 F 等价。

[例 2] 考察 6.1 节中的关系模式 $S < U, F >$,其中:

$U = \{\text{Sno}, \text{Sdept}, \text{Mname}, \text{Cno}, \text{Grade}\}$,

$F = \{\text{Sno} \rightarrow \text{Sdept}, \text{Sdept} \rightarrow \text{Mname}, (\text{Sno}, \text{Cno}) \rightarrow \text{Grade}\}$

设 $F' = \{\text{Sno} \rightarrow \text{Sdept}, \text{Sno} \rightarrow \text{Mname}, \text{Sdept} \rightarrow \text{Mname}, (\text{Sno}, \text{Cno}) \rightarrow \text{Grade}, (\text{Sno}, \text{Sdept}) \rightarrow \text{Sdept}\}$

根据**定义 6.15** 可以验证 F 是最小覆盖,而 F' 不是。因为 $F' - \{\text{Sno} \rightarrow \text{Mname}\}$ 与 F' 等价,$F' - \{(\text{Sno}, \text{Sdept}) \rightarrow \text{Sdept}\}$ 与 F' 等价。

定理 6.3 每一个函数依赖集 F 均等价于一个极小函数依赖集 F_m。此 F_m 称为 F 的最小依赖集。

证 这是一个构造性的证明,分三步对 F 进行"极小化处理",找出 F 的一个最小依赖集来。

(1) 逐一检查 F 中各函数依赖 $FD_i: X \rightarrow Y$,若 $Y = A_1 A_2 \cdots A_k$,$k > 2$,则用 $\{X \rightarrow A_j \mid j = 1, 2, \cdots, k\}$ 来取代 $X \rightarrow Y$。

(2) 逐一检查 F 中各函数依赖 $FD_i: X \rightarrow A$,令 $G = F - \{X \rightarrow A\}$,若 $A \in X_G^+$,则从 F 中去掉此函数依赖(因为 F 与 G 等价的充要条件是 $A \in X_G^+$)。

(3) 逐一取出 F 中各函数依赖 $FD_i: X \rightarrow A$,设 $X = B_1 B_2 \cdots B_m$,逐一考查 B_i ($i = 1, 2, \cdots, m$),若 $A \in (X - B_i)_F^+$,则以 $X - B_i$ 取代 X(因为 F 与 $F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ 等价的充要条件是 $A \in Z_F^+$ 其中 $Z = X - B_i$)。

最后剩下的 F 就一定是极小依赖集,并且与原来的 F 等价。因为对 F 的每一次"改造"都保证了改造前后的两个函数依赖集等价。这些证明很显然,请读者自行补上。

应当指出,F 的最小依赖集 F_m 不一定是唯一的,它与对各函数依赖 FD_i 及 $X \rightarrow A$ 中 X 各属性的处置顺序有关。

[例 3] $F = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, A \rightarrow C, C \rightarrow A\}$

$F_{m1} = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

$F_{m2} = \{A \rightarrow B, B \rightarrow A, A \rightarrow C, C \rightarrow A\}$

这里给出了 F 的两个最小依赖集 F_{m1}、F_{m2}。

若改造后的 F 与原来的 F 相同，说明 F 本身就是一个最小依赖集，因此**定理 6.3** 的证明给出的极小化过程也可以看成是检验 F 是否为极小依赖集的一个算法。

两个关系模式 $\boldsymbol{R}_1<U,F>$，$\boldsymbol{R}_2<U,G>$，如果 F 与 G 等价，那么 $\boldsymbol{R}_1$ 的关系一定是 $\boldsymbol{R}_2$ 的关系。反过来，$\boldsymbol{R}_2$ 的关系也一定是 $\boldsymbol{R}_1$ 的关系。所以在 $\boldsymbol{R}<U,F>$ 中用与 F 等价的依赖集 G 来取代 F 是允许的。

* 6.4 模式的分解

在对函数依赖的基本性质有了初步的了解之后，可以具体地来讨论模式的分解了。

定义 6.16 关系模式 $\boldsymbol{R}<U,F>$ 的一个分解是指 $\rho=\{\boldsymbol{R}_1<U_1,F_1>,\boldsymbol{R}_2<U_2,F_2>,\cdots,\boldsymbol{R}_n<U_n,F_n>\}$

其中 $U=\bigcup_{i=1}^{n}U_i$，并且没有 $U_i\subseteq U_j,1\leqslant i,j\leqslant n$，$F_i$ 是 F 在 U_i 上的投影。

所谓“F_i 是 F 在 U_i 上的投影”的确切定义是：

定义 6.17 函数依赖集合 $\{X\rightarrow Y|X\rightarrow Y\in F^{+}\wedge XY\subseteq U_i\}$ 的一个覆盖 F_i 叫作 F 在属性 U_i 上的投影。

6.4.1 模式分解的 3 个定义

对于一个模式的分解是多种多样的，但是分解后产生的模式应与原模式等价。

人们从不同的角度去观察问题，对“等价”的概念形成了三种不同的定义：

- 分解具有“无损连接性”(Lossless join)；
- 分解要“保持函数依赖”(Preserve dependency)；
- 分解既要“保持函数依赖”，又要具有“无损连接性”。

这 3 个定义是实行分解的 3 条不同的准则。按照不同的分解准则，模式所能达到的分离程度各不相同，各种范式就是对分离程度的测度。

这一节要讨论的问题是：

(1) “无损连接性”和“保持函数依赖”的含义是什么？如何判断？

(2) 对于不同的分解等价定义，究竟能达到何种程度的分离，即分离后的关系模式是第几范式；

(3) 如何实现分离，即给出分解的算法。

先来看两个例子，说明按**定义 6.16**，若只要求 $\boldsymbol{R}<U,F>$ 分解后的各关系模式所含属性的“并”等于 U，这个限定是很不够的。

一个关系分解为多个关系，相应地原来存储在一张二维表内的数据就要分散存储到多张二维表中，要使这个分解有意义，起码的要求是后者不能丢失前者的信息。

[例4] 已知关系模式 $R<U,F>$，其中 $U=\{\text{Sno},\text{Sdept},\text{Mname}\}$，$F=\{\text{Sno}\rightarrow\text{Sdept},\text{Sdept}\rightarrow\text{Mname}\}$。$R<U,F>$ 的元组语义是学生 Sno 正在 Sdept 系学习，其系主任是 Mname。并且一个学生(Sno)只在一个系学习，一个系只有一名系主任。R 的一个关系见表6.5所示。

由于 R 中存在传递函数依赖 Sno→Mname，它会发生更新异常。例如，如果 S4 毕业，则 D3 系的系主任是王一的信息也就丢掉了。反过来，如果一个系 D5 尚无在校学生，那么这个系的系主任是赵某的信息也无法存入。于是进行了如下分解：

$$\rho_1=\{R_1<\text{Sno},\phi>,R_2<\text{Sdept},\phi>,R_3<\text{Mname},\phi>\}$$

表6.5 R 的一个关系示例

Sno	Sdept	Mname
S1	D1	张五
S2	D1	张五
S3	D2	李四
S4	D3	王一

分解后诸 R_i 的关系 r_i 是 R 在 U_i 上的投影，即 $r_i=R[U_i]$

$r_1=\{\text{S1},\text{S2},\text{S3},\text{S4}\}$，$r_2=\{\text{D1},\text{D2},\text{D3}\}$，$r_3=\{\text{张五},\text{李四},\text{王一}\}$。

对于分解后的数据库，要回答“S1 在哪个系学习”也不可能了。这样的分解还有什么用呢。

如果分解后的数据库能够恢复到原来的情况，不丢失信息的要求也就达到了。R_i 向 R 的恢复是通过自然连接来实现的，这就产生了无损连接性的概念。显然，本例的分解 ρ_1 所产生的诸关系自然连接的结果实际上是它们的笛卡尔积，元组增加了，信息丢失了。

于是对 R 又进行另一种分解：

$$\rho_2=\{R_1<\{\text{Sno},\text{Sdept}\},\{\text{Sno}\rightarrow\text{Sdept}\}>,R_2<\{\text{Sno},\text{Mname}\},\{\text{Sno}\rightarrow\text{Mname}\}>\}$$

以后可以证明 ρ_2 对 R 的分解是可恢复的，但是前面提到的插入和删除异常仍然没有解决，原因就在于原来在 R 中存在的函数依赖 Sdept→Mname，现在在 R_1 和 R_2 中都不再存在了。因此人们又要求分解具有“保持函数依赖”的特性。

最后对 R 进行了以下一种分解：

$$\rho_3=\{R_1<\{\text{Sno},\text{Sdept}\},\{\text{Sno}\rightarrow\text{Sdept}\}>,R_2<\{\text{Sdept},\text{Mname}\},\{\text{Sdept}\rightarrow\text{Mname}\}>\}$$

可以证明分解 ρ_3 既具有无损连接性，又保持函数依赖。它解决了更新异常，又没有丢失原数据库的信息，这是所希望的分解。

由此，可以看出为什么要提出对数据库模式“等价”的3个不同定义的原因。

下面严格地定义分解的无损连接性和保持函数依赖性并讨论它们的判别算法。

6.4.2 分解的无损连接性和保持函数依赖性

先定义一个记号：设 $\rho=\{R_1<U_1,F_1>,\cdots,R_k<U_k,F_k>\}$ 是 $R<U,F>$ 的一个分解，r

是 $R<U,F>$ 的一个关系。定义 $m_\rho(r)=\bowtie_{i=1}^{k}\pi_{R_i}(r)$，即 $m_\rho(r)$ 是 r 在 ρ 中各关系模式上投影的连接。这里 $\pi_{R_i}(r)=|t.U_i|t\in r|$。

引理 6.4 设 $R<U,F>$ 是一个关系模式，$\rho=|R_1<U_1,F_1>,\cdots,R_k<U_k,F_k>|$ 是 R 的一个分解，r 是 R 的一个关系，$r_i=\pi_{R_i}(r)$，则

(1) $r\subseteq m_\rho(r)$

(2) 若 $s=m_\rho(r)$，则 $\pi_{R_i}(s)=r_i$

(3) $m_\rho(m_\rho(r))=m_\rho(r)$

证：

(1) 证明 r 中的任何一个元组属于 $m_\rho(r)$。

任取 r 中的一个元组 t，$t\in r$，设 $t_i=t.U_i(i=1,2,\cdots,k)$。对 k 进行归纳可以证明 $t_1t_2\cdots t_k\in\bowtie_{i=1}^{k}\pi_{R_i}(r)$，所以 $t\in m_\rho(r)$，即 $r\subseteq m_\rho(r)$。

(2) 由(1)得到 $r\subseteq m_\rho(r)$，已设 $s=m_\rho(r)$，所以，$r\subseteq s$，$\pi_{R_i}(r)\subseteq\pi_{R_i}(s)$ 现只需证明 $\pi_{R_i}(s)\subseteq\pi_{R_i}(r)$，就有 $\pi_{R_i}(s)=\pi_{R_i}(r)=r_i$。

任取 $S_i\in\pi_{R_i}(s)$，必有 S 中的一个元组 v，使得 $v.U_i=S_i$。根据自然连接的定义 $v=t_1t_2\cdots t_k$，对于某中每一个 t_i 必存在 r 中的一个元组 t，使得 $t.U_i=t_i$。由前面 $\pi_{R_i}(r)$ 的定义即得 $t_i\in\pi_{R_i}(r)$。又因 $v=t_1t_2\cdots t_k$，故 $v.U_i=t_i$。又由上面证得：$v.U_i=S_i$，$t_i\in\pi_{R_i}(r)$，故 $S_i\in\pi_{R_i}(r)$。即 $\pi_{R_i}(s)\subseteq\pi_{R_i}(r)$。故 $\pi_{R_i}(s)=\pi_{R_i}(r)$。

(3) $m_\rho(m_\rho(r))=\bowtie_{i=1}^{k}\pi_{R_i}(m_\rho(r))=\bowtie_{i=1}^{k}\pi_{R_i}(s)=\bowtie_{i=1}^{k}\pi_{R_i}(r)=m_\rho(r)$

定义 6.18 $\rho=|R_1<U_1,F_1>,\cdots,R_k<U_k,F_k>|$ 是 $R<U,F>$ 的一个分解，若对 $R<U,F>$ 的任何一个关系 r 均有 $r=m_\rho(r)$ 成立，则称分解 ρ 具有无损连接性。简称 ρ 为无损分解。

直接根据**定义 6.18** 去鉴别一个分解的无损连接性是不可能的，**算法 6.2** 给出了一个判别的方法。

算法 6.2 判别一个分解的无损连接性。

$\rho=|R_1<U_1,F_1>,\cdots,R_k<U_k,F_k>|$ 是 $R<U,F>$ 的一个分解，$U=|A_1,\cdots,A_n|$，$F=|FD_1,FD_2,\cdots,FD_p|$，不妨设 F 是一极小依赖集，记 FD_i 为 $X_i\to A_{i_1}$。

(1) 建立一张 n 列 k 行的表。每一列对应一个属性，每一行对应分解中的一个关系模式。若属性 A_j 属于 U_i，则在 j 列 i 行交叉处填上 a_j，否则填上 b_{ij}；

(2) 对每一个 FD_i 做下列操作：找到 X_i 所对应的列中具有相同符号的那些行。考察这些行中 l_i 列的元素，若其中有 a_{li}，则全部改为 a_{li}；否则全部改为 b_{mli}；m 是这些行的行号最小值。

应当注意的是，若某个 b_{di} 被更动，那么该表的 li 列中凡是 b_{di} 的符号(不管它是否开始找到的那些行)均应作相应的更改。

如在某次更改之后，有一行成为 $a_1,a_2,\cdots,a_n$。则算法终止。ρ 具有无损连接性，否则 ρ 不具有无损连接性。

对 F 中 p 个 FD 逐一进行一次这样的处置，称为对 F 的一次扫描。

(3) 比较扫描前后，表有无变化。如有变化，则返回第(2)步，否则算法终止。

如果发生循环，那么前次扫描至少应使该表减少一个符号，表中符号有限，因此循环必然终止。

定理 6.4 如果**算法 6.2** 终止时，表中有一行为 $a_1,a_2,\cdots,a_n$，则 ρ 为无损连接分解。

证明从略。

[**例 5**] 已知 $\boldsymbol{R}<U,F>$，$U=\{A,B,C,D,E\}$，$F=\{AB\rightarrow C,C\rightarrow D,D\rightarrow E\}$，$\boldsymbol{R}$ 的一个分解为 $\boldsymbol{R}_1(A,B,C)$，$\boldsymbol{R}_2(C,D)$，$\boldsymbol{R}_3(D,E)$。

(1) 首先构造初始表，如图 6.9(a)。

(2) 对 $AB\rightarrow C$，因各元组的第一、二列没有相同的分量，所以表不改变。由 $C\rightarrow D$ 可以把 b_{14} 改为 a_4，再由 $D\rightarrow E$ 可使 b_{15},b_{25} 全改为 a_5。最后结果为图 6.9(b)。表中第一行成为 a_1,a_2,a_3,a_4,a_5，所以此分解具有无损连接性。

当关系模式 $\boldsymbol{R}$ 分解为两个关系模式 $\boldsymbol{R}_1,\boldsymbol{R}_2$ 时有下面的判定准则。

定理 6.5 对于 $\boldsymbol{R}<U,F>$ 的一个分解 $\rho=\{\boldsymbol{R}_1<U_1,F_1>\boldsymbol{R}_2<U_2,F_2>\}$，如果 $U_1\cap U_2\rightarrow U_1-U_2\in F^+$ 或 $U_1\cap U_2\rightarrow U_2-U_1\in F^+$，则 ρ 具有无损连接性。

定理的证明留给读者完成。

定义 6.19 若 $F^+=(\bigcup_{i=1}^{k}F_i)^+$，则 $\boldsymbol{R}<U,F>$ 的分解 $\rho=\{\boldsymbol{R}_1<U_1,F_1>,\cdots,\boldsymbol{R}_k<U_k,F_k>\}$ 保持函数依赖。

引理 6.3 给出了判断两个函数依赖集等价的可行算法。因此**引理 6.3** 也给出了判别 $\boldsymbol{R}$ 的分解 ρ 是否保持函数依赖的方法。

A	B	C	D	E
a_1	a_2	a_3	b_{14}	b_{15}
b_{21}	b_{22}	a_3	a_4	b_{25}
b_{31}	b_{32}	b_{33}	a_4	a_5

(a)

A	B	C	D	E
a_1	a_2	a_3	a_4	a_5
b_{21}	b_{22}	a_3	a_4	a_5
b_{31}	b_{32}	b_{33}	a_4	a_5

(b)

图 6.9 分解具有无损连接的一个实例

6.4.3 模式分解的算法

关于模式分解的几个重要事实是：

(1) 若要求分解保持函数依赖，那么模式分离总可以达到3NF，但不一定能达到BCNF；

(2) 若要求分解既保持函数依赖，又具有无损连接性，可以达到3NF，但不一定能达到BCNF；

(3) 若要求分解具有无损连接性，那一定可达到4NF。

它们分别由**算法6.3**、**算法6.4**、**算法6.5**、**算法6.6**来实现。

算法6.3（合成法）转换为3NF的保持函数依赖的分解。

(1) 对$\boldsymbol{R}<U,F>$中的函数依赖集F进行"极小化处理"（处理后得到的依赖集仍记为F）。

(2) 找出不在F中出现的属性，把这样的属性构成一个关系模式。把这些属性从U中去掉，剩余的属性仍记为U。

(3) 若有$X\rightarrow A\in F$，且$XA=U$，则$\rho=\{\boldsymbol{R}\}$，算法终止。

(4) 否则，对F按具有相同左部的原则分组（假定分为k组），每一组函数依赖F_i'所涉及的全部属性形成一个属性集U_i。若$U_i\subseteq U_j(i\neq j)$就去掉$U_i$。由于经过了步骤(2)，故$U=\bigcup_{i=1}^{k}U_i$，于是$\rho=\{\boldsymbol{R}_1<U_1,F_1>,\cdots,\boldsymbol{R}_k<U_k,F_k>\}$构成$\boldsymbol{R}<U,F>$的一个保持函数依赖的分解，并且每个$\boldsymbol{R}_i<U_i,F_i>$均属3NF。这里$F_i$是$F$在$U_i$上的投影，并且$F_i$不一定与$F_i'$相等，但$F_i'$一定被$F_i$所包含，因此分解$\rho$保持函数依赖是显然的。

下面证明每一个$\boldsymbol{R}_i<U_i,F_i>$一定属于3NF。

设$F_i'=\{X\rightarrow A_1,X\rightarrow A_2,\cdots,X\rightarrow A_k\}$，$U_i=\{X,A_1,A_2,\cdots,A_k\}$

(1) $\boldsymbol{R}_i<U_i,F_i>$一定以$X$为码。

(2) 若$\boldsymbol{R}_i<U_i,F_i>$不属于3NF，则必存在非主属性$A_m(l\leqslant m\leqslant k)$及属性组合$Y$，$A_m\notin Y$，使得$X\rightarrow Y$，$Y\rightarrow A_m\in F_i^+$，而$Y\rightarrow X\notin F_i^+$。

若$Y\subset X$，则与$X\rightarrow A_m$属于最小依赖集F相矛盾，因而$Y\not\subseteq X$。不妨设$Y\cap X=X_1$，$Y-X=\{A_1,\cdots,A_p\}$，令$G=F-\{X\rightarrow A_m\}$，显然$Y\subseteq X_G^+$，即$X\rightarrow Y\in G^+$。

可以断言$Y\rightarrow A_m$也属于G^+。因为$Y\rightarrow A_m\in F_i^+$，所以$A_m\in Y_F^+$。若$Y\rightarrow A_m$不属于$G^+$，则在求$Y_F^+$的算法中，只有使用$X\rightarrow A_m$才能将$A_m$引入。于是按**算法6.1**必有$j$，使得$X\subseteq Y^{(j)}$，于是$Y\rightarrow X$成立是矛盾的。

于是$X\rightarrow A_m$属于G^+，与F是最小依赖集相矛盾。所以$\boldsymbol{R}_i<U_i,F_i>$一定属于3NF。

算法6.4 转换为3NF既有无损连接性又保持函数依赖的分解。

(1) 设X是$\boldsymbol{R}<U,F>$的码。$\boldsymbol{R}<U,F>$已由**算法6.3**分解为$\rho=\{\boldsymbol{R}_1<U_1,F_1>,\boldsymbol{R}_2<U_2,F_2>,\cdots,\boldsymbol{R}_k<U_k,F_k>\}$，令$\tau=\rho\cup\{\boldsymbol{R}^*<X,F_x>\}$。

(2) 若有某个U_i，$X\subseteq U_i$，将$\boldsymbol{R}^*<X,F_x>$从τ中去掉，

(3) τ 就是所求的分解。

$\boldsymbol{R}^* < X, F_x >$ 显然属于3NF，而 τ 保持函数依赖也很显然，只要判定 τ 的无损连接性就行了。

由于 τ 中必有某关系模式 $\boldsymbol{R}(T)$ 的属性组 $T \supseteq X$。由于 X 是 $\boldsymbol{R} < U, F >$ 的码，任取 $U - T$ 中的属性 B，必存在某个 i，使 $B \in T^{(i)}$（按**算法 6.1**）。对 i 施行归纳法可以证明由**算法 6.2**，表中关系模式 $\boldsymbol{R}(T)$ 所在的行一定可成为 $a_1, a_2, \cdots, a_n$。τ 的无损连接性得证。

算法 6.5　(分解法)转换为 BCNF 的无损连接分解。

(1) 令 $\rho = \{\boldsymbol{R} < U, F >\}$

(2) 检查 ρ 中各关系模式是否均属于 BCNF。若是，则算法终止。

(3) 设 ρ 中 $R_i < U_i, F_i >$ 不属于 BCNF，那么必有 $X \rightarrow A \in F_i^+$ $(A \notin X)$，且 X 非 $\boldsymbol{R}_i$ 的码。因此，XA 是 U_i 的真子集。对 $\boldsymbol{R}_i$ 进行分解：$\sigma = \{S_1, S_2\}$，$U_{S1} = XA$，$U_{S2} = U_i - \{A\}$，以 σ 代替 $\boldsymbol{R}_i(U_i, F_i)$ 返回第(2)步。

由于 U 中属性有限，因而有限次循环后**算法 6.5** 一定会终止。

这是一个自顶向下的算法。它自然地形成一棵对 $\boldsymbol{R}(U, F)$ 的二叉分解树。应当指出，$\boldsymbol{R} < U, F >$ 的分解树不一定是唯一的。这与步骤(3)中具体选定的 $X \rightarrow A$ 有关。

算法 6.5 最初令 $\rho = \{\boldsymbol{R} < U, F >\}$，显然 ρ 是无损连接分解，而以后的分解则由下面的**引理 6.5** 保证了它的无损连接性。

引理 6.5　若 $\rho = \{\boldsymbol{R}_1 < U_1, F_1 >, \cdots, \boldsymbol{R}_k < U_k, F_k >\}$ 是 $\boldsymbol{R} < U, F >$ 的一个无损连接分解，$\sigma = \{S_1, S_2, \cdots, S_m\}$ 是 ρ 中 $R_i < U_i, F_i >$ 的一个无损连接分解，那么

$$\rho' = \{\boldsymbol{R}_1, \boldsymbol{R}_2, \cdots, \boldsymbol{R}_{i-1}, S_1, \cdots, S_m, \boldsymbol{R}_{i+1}, \cdots, \boldsymbol{R}_k\},$$

ρ'' = $\{\boldsymbol{R}_1, \cdots, \boldsymbol{R}_k, \boldsymbol{R}_{k+1}, \cdots, \boldsymbol{R}_n\}$ (ρ''是 $\boldsymbol{R} < U, F >$ 包含 ρ 的关系模式集合的分解)，均是 $\boldsymbol{R} < U, F >$ 的无损连接分解。

证明的关键是自然连接的结合律，给出结合律的证明，其他部分留给读者。

引理 6.6　$(\boldsymbol{R}_1 \bowtie \boldsymbol{R}_2) \bowtie \boldsymbol{R}_3 = \boldsymbol{R}_1 \bowtie (\boldsymbol{R}_2 \bowtie \boldsymbol{R}_3)$

证

设 r_i 是 $\boldsymbol{R}_i < U_i, F_i >$ 的关系，$i = 1, 2, 3$

设 $U_1 \cap U_2 \cap U_3 = V$;

$U_1 \cap U_2 - V = X$;

$U_2 \cap U_3 - V = Y$;

$U_1 \cap U_3 - V = Z$(如图 6.10 所示)。

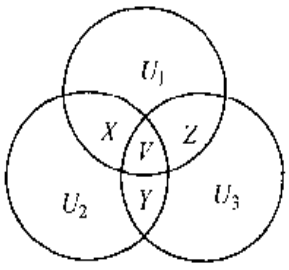


图 6.10　引理 6.6 三个关系属性的示意图

容易证明 t 是 $(\boldsymbol{R}_1 \bowtie \boldsymbol{R}_2) \bowtie \boldsymbol{R}_3$ 中的一个元组的充要条件是：

T_{R_1}、T_{R_2}、T_{R_3} 是 t 的连串，这里 $T_{R_i} \in r_i$ $(i = 1, 2, 3)$，$T_{R_1}[V] = T_{R_2}[V] = T_{R_3}[V]$，$T_{R_1}[X] = T_{R_2}[X]$，$T_{R_1}[Z] = T_{R_3}[Z]$，$T_{R_2}[Y] = T_{R_3}[Y]$。而这也是 t 为 $\boldsymbol{R}_1 \bowtie (\boldsymbol{R}_2 \bowtie \boldsymbol{R}_3)$ 中的元组的充要条件。于是有

$$(\boldsymbol{R}_1 \bowtie \boldsymbol{R}_2) \bowtie \boldsymbol{R}_3 = \boldsymbol{R}_1 \bowtie (\boldsymbol{R}_2 \bowtie \boldsymbol{R}_3)$$

在6.2.8节中已经指出，一个关系模式中若存在多值依赖(指非平凡的非函数依赖的多值依赖)，则数据的冗余度大而且存在插入、修改、删除异常等问题。为此要消除这种多值依赖，使模式分离达到一个新的高度4NF。下面讨论达到4NF的具有无损连接性的分解。

定理6.6 关系模式 $\boldsymbol{R}<U,D>$ 中，D 为 $\boldsymbol{R}$ 中函数依赖FD和多值依赖MVD的集合。

则 $X\rightarrow\rightarrow Y$ 成立的充要条件是 $\boldsymbol{R}$ 的分解 $\rho=\{\boldsymbol{R}_1(X,Y),\boldsymbol{R}_2(X,Z)\}$ 具有无损连接性，其中 $Z=U-X-Y$。

证 先证充分性。

若 ρ 是 $\boldsymbol{R}$ 的一个无损连接分解，则对 $\boldsymbol{R}<U,F>$ 的任一关系 r 有：

$$r=\pi_{R_1}(r)\bowtie\pi_{R_2}(r)$$

设 $t,s\in r$，且 $t[X]=s[X]$，于是 $t[XY],s[XY]\in\pi_{R_1}(r)$，$t[XZ],s[XZ]\in\pi_{R_2}(r)$。由于 $t[X]=s[X]$，所以 $t[XY]\cdot s[XZ]$ 与 $t[XZ]\cdot s[XY]$ 均属于 $\pi_{R_1}(r)\bowtie\pi_{R_2}(r)$，也即属于 r。令 $u=t[XY]\cdot s[XZ]$，$v=t[XZ]\cdot s[XY]$，就有 $u[X]=v[X]=t[X]$，$u[Y]=t[Y]$，$u[Z]=s[Z]$，$v[Y]=s[Y]$，$v[Z]=t[Z]$，所以 $X\rightarrow\rightarrow Y$ 成立。

再证必要性。

若 $X\rightarrow\rightarrow Y$ 成立，对于 $\boldsymbol{R}<U,D>$ 的任一关系 r，任取 $\omega\in\pi_{R_1}(r)\bowtie\pi_{R_2}(r)$，则必有 $t,s\in r$，使得 $\omega=t[XY]\cdot s[XZ]$，由于 $X\rightarrow\rightarrow Y$ 对 $\boldsymbol{R}<U,D>$ 成立，ω 应当属于 r，所以 ρ 是无损连接分解。

定理6.6 给出了对 $\boldsymbol{R}<U,D>$ 的一个无损的分解方法。若 $\boldsymbol{R}<U,D>$ 中 $X\rightarrow\rightarrow Y$ 成立，则 $\boldsymbol{R}$ 的分解 $\rho=\{\boldsymbol{R}_1(X,Y),\boldsymbol{R}_2(X,Z)\}$ 具有无损连接性。

算法6.6 达到4NF的具有无损连接性的分解。

首先使用**算法6.5**，得到 $\boldsymbol{R}$ 的一个达到了BCNF的无损连接分解 ρ。然后对某一 $\boldsymbol{R}_i<U_i,D_i>$，若不属于4NF，则可按**定理6.6**的作法进行分解。直到每一个关系模式均属于4NF为止。**定理6.6**和**引理6.5**保证了最后得到的分解的无损连接性。

关系模式 $\boldsymbol{R}<U,D>$，U 是属性总体集，D 是 U 上的一组数据依赖(函数依赖和多值依赖)，对于包含函数依赖和多值依赖的数据依赖有一个有效且完备的公理系统。

A1：若 $Y\subseteq X\subseteq U$，则 $X\rightarrow Y$；

A2：若 $X\rightarrow Y$，且 $Z\subseteq U$，则 $XZ\rightarrow YZ$；

A3：若 $X\rightarrow Y$，$Y\rightarrow Z$，则 $X\rightarrow Z$；

A4：若 $X\rightarrow\rightarrow Y$，$V\subseteq W\subseteq U$，则 $XW\rightarrow\rightarrow YV$；

A5：若 $X\rightarrow\rightarrow Y$，则 $X\rightarrow\rightarrow U-X-Y$；

A6：若 $X\rightarrow\rightarrow Y$，$Y\rightarrow\rightarrow Z$，则 $X\rightarrow\rightarrow Z-Y$；

A7：若 $X\rightarrow Y$，则 $X\rightarrow\rightarrow Y$；

A8：若 $X\rightarrow\rightarrow Y$，$W\rightarrow Z$，$W\cap Y=\phi$，$Z\subseteq Y$，则 $X\rightarrow Z$。

公理系统的有效性是指从 D 出发根据 8 条公理推导出的函数依赖或多值依赖一定为 D 蕴含;完备性是指凡 D 所蕴含的函数依赖或多值依赖均可以从 D 根据 8 条公理推导出来。也就是说,在函数依赖和多值依赖的条件下,"蕴含"与"导出"仍是等价的。

A1、A2、A3 的有效性前面 6.3 节已证明,其余的有效性证明,留给读者。

由 8 条公理可得如下 4 条有用的推理规则:

(1) 合并规则:$X\rightarrow\rightarrow Y, X\rightarrow\rightarrow Z$,则 $X\rightarrow\rightarrow YZ$;

(2) 伪传递规则:$X\rightarrow\rightarrow Y, WY\rightarrow Z$,则 $WX\rightarrow\rightarrow Z-WY$;

(3) 混合伪传递规则:$X\rightarrow\rightarrow Y, XY\rightarrow Z$,则 $X\rightarrow Z-Y$;

(4) 分解规则:$X\rightarrow\rightarrow Y, X\rightarrow Z$,则 $X\rightarrow\rightarrow Y\cap Z, X\rightarrow\rightarrow Y-Z, X\rightarrow\rightarrow Z-Y$。

6.5 小　结

在函数依赖、多值依赖的范畴内讨论了关系模式的规范化,其基本思想可用图 6.11 表示。

在整个讨论过程中,只采用了两种关系运算——投影和自然连接。并且总是从一个关系模式出发,而不是从一组关系模式出发实行分解。"等价"的定义也是一组关系模式与一个关系模式的"等价",这就是说,在开始讨论问题时事实上已经假设了存在着一个单一的关系模式,这就是泛关系假设。

本章一开始,就指出这是研究模式设计的一种特殊情况:"假设已知一个模式 Sϕ,它仅由单个关系模式组成,问题是要设计一个模式 SD,它与 Sϕ'等价',但在某些方面更好一些"。

泛关系假设是运用规范化理论时的障碍,因为承认了泛关系假设,就等于承认了现实世界各实体间只能有一种联系,而这常常是办不到的。比如工人与机器之间就可以存在"使

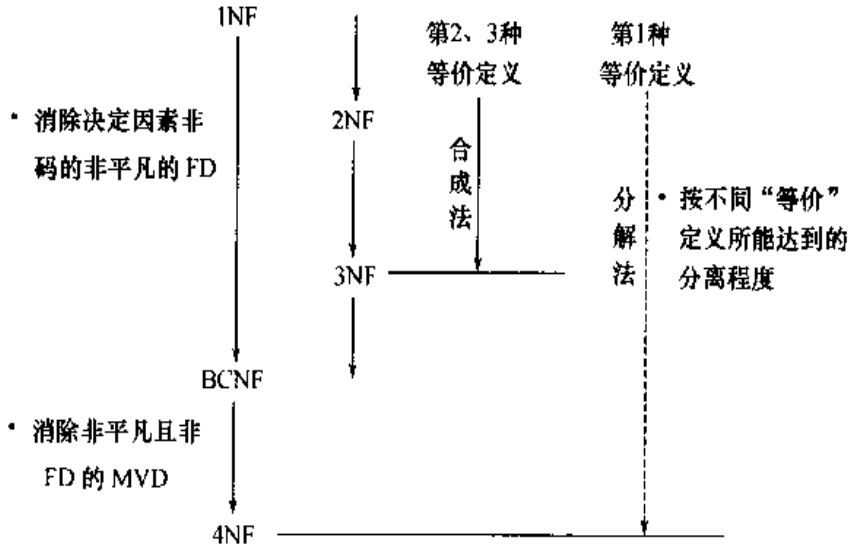


图 6.11　关系模式规范化小结

用”、“维护”等多种联系。对此，人们提出了一些办法，希望解决这个矛盾。例如建立一个不受泛关系假设限制的理论，或者采用某些手段使现实世界向信息世界转换时适合于泛关系的要求。

最后应当强调的是，规范化理论为数据库设计提供了理论的指南和工具，但仅仅是指南和工具。并不是规范化程度越高，模式就越好，而必须结合应用环境和现实世界的具体情况合理地选择数据库模式。

习　题

1. 理解并给出下列术语的定义：

函数依赖、部分函数依赖、完全函数依赖、传递依赖、候选码、主码、外码、全码(All-key)、1NF、2NF、3NF、BCNF、多值依赖、4NF。

2. 建立一个关于系、学生、班级、学会等诸信息的关系数据库。

描述学生的属性有：学号、姓名、出生年月、系名、班号、宿舍区；

描述班级的属性有：班号、专业名、系名、人数、入校年份；

描述系的属性有：系名、系号、系办公室地点、人数；

描述学会的属性有：学会名、成立年份、地点、人数。

有关语义如下：一个系有若干专业，每个专业每年只招一个班，每个班有若于学生。一个系的学生住在同一宿舍区。每个学生可参加若干学会，每个学会有若干学生。学生参加某学会有一个入会年份。

请给出关系模式，写出每个关系模式的极小函数依赖集，指出是否存在传递函数依赖，对于函数依赖左部是多属性的情况讨论函数依赖是完全函数依赖，还是部分函数依赖。

指出各关系的候选码、外部码，有没有全码存在？

*3. 试由 Armostrong 公理系统推导出下面 3 条推理规则：

(1) 合并规则：若 $X\to Z, X\to Y$，则有 $X\to YZ$

(2) 伪传递规则：由 $X\to Y, WY\to Z$ 有 $XW\to Z$

(3) 分解规则：$X\to Y, Z\subseteq Y$，有 $X\to Z$

4. 关于多值依赖的另一种定义是：

给定一个关系模式 $R(X,Y,Z)$，其中 X、Y、Z 可以是属性或属性组合。

设 $x\in X, y\in Y, z\in Z$，xz 在 R 中的像集为

$$Y_{xz}=\{r.Y \mid r.X=x\wedge r.Z=z\wedge r\in R\}$$

定义 $R(X,Y,Z)$ 当且仅当 $Y_{xz}=Y_{xz'}$ 对于每一组 (x,z,z') 都成立，则 Y 对 X 多值依赖，记作 $X\twoheadrightarrow Y$。这里，允许 Z 为空集，在 Z 为空集时，称为平凡的多值依赖。

请证明这里的定义和 6.2.7 节中**定义 6.9** 是等价的。

5. 试举出 3 个多值依赖的实例。

*6. 试证明书上给出的关于 FD 和 MVD 公理系统的 A_4、A_5 和 A_6。

*7. 设关系模式为 $R(U,F)$，X,Y 为属性集，$X,Y\subseteq U$。

证明：(1) $X\subseteq X_F^+$

(2) $(X_F^+)_F^+ = X_F^+$

(3) 若 $X \subseteq Y$ 则 $X_F^+ \subseteq Y_F^+$

(4) $U_F^+ = U$

*8. 设关系模式为 $R(U,F)$,若 $X_F^+ = X$,则称 X 相对于 F 是饱和的。定义饱和集 $\varphi_F = \{X \mid X = X_F^+\}$,试证明 $\varphi_F = \{X_F^+ \mid X \subseteq U\}$。

*9. 图 6.12 表示一个公司各部门的层次结构。

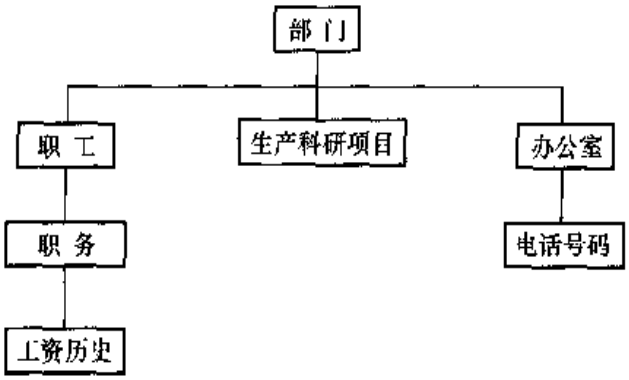


图 6.12 某公司的层次结构

对每个部门,数据库中包含部门号(唯一的)D#、预算费(BUGET)以及此部门领导人员的职工号 E#(唯一的)信息。

对每一个部门,还存有关于此部门的全部职工、生产与科研项目以及办公室的信息。

职工信息包括:职工号、他所参加的生产与科研项目号(J#)、他所在办公室的电话号码(PHONE#)。

生产科研项目包含:项目号(唯一的)、预算费。

办公室信息包含办公室房间号(唯一的)、面积。

对每个职工,数据库中有他曾担任过的职务以及担任某一职务时的工资历史。

对每个办公室包含此办公室中全部电话号码的信息。

请给出你认为合理的数据依赖,把这个层次结构转换成一组规范化的关系。

提示:此题可分步完成,第一步先转换成一组 1NF 的关系,然后逐步转换为 2NF,3NF,…。

*10. 在一个订货系统的数据库中,存有顾客、货物和订货单的信息。

每个顾客包含顾客号(唯一的)、收货地址(一个顾客可有几个地址)、赊购限额、余额以及折扣。

每个订货单包含顾客号、收货地址、订货日期、订货细则(每个订货单有若干条)、每条订货细则内容为货物号以及订货数量。

每种货物包含货物号(唯一的)、制造厂商、每个厂商的实际存货量、规定的最低存货量和货物描述。

由于处理上的要求,每个订货单的每一订货细则中还应有一个未发货量(此值初始时为订货数量,随着发货将减为零)。

为这些数据设计一个数据库,如前题那样,给出认为合理的数据依赖。

*11. 设在第 10 题中实际上只有很少量的顾客(例如 1%)有多个发货地址,由于这些少数的而又不能忽视的情形使得不能按一般的方式来处理问题。你能发现第 10 题答案中的问题吗?能设法改进吗?

12. 下面的结论哪些是正确的?哪些是错误的?对于错误的请给出一个反例说明之。

(1) 任何一个二目关系是属于3NF的。

(2) 任何一个二目关系是属于BCNF的。

(3) 任何一个二目关系是属于4NF的。

(4) 当且仅当函数依赖 $A\rightarrow B$ 在 R 上成立,关系 $R(A,B,C)$ 等于其投影 $R_1(A,B)$ 和 $R_2(A,C)$ 的连接。

(5) 若 $R.A\rightarrow R.B, R.B\rightarrow R.C$,则 $R.A\rightarrow R.C$

(6) 若 $R.A\rightarrow R.B, R.A\rightarrow R.C$,则 $R.A\rightarrow R.(B,C)$

(7) 若 $R.B\rightarrow R.A, R.C\rightarrow R.A$,则 $R.(B,C)\rightarrow R.A$

(8) 若 $R.(B,C)\rightarrow R.A$,则 $R.B\rightarrow R.A, R.C\rightarrow R.A$

本章参考文献

1 Codd E F. Normalized Data Base Structure: A Brief Tutorial. Proc. of ACM SIGFIDET Workshop on Data Description, Access and Control. 1971

2 Codd E F. Further Normalized Data Base Relational Model. In Data Base Systems, Courant Computer Science Symposia Series. Vol. 6, Englewood Ciffs. N. J. Prentice-Hall, 1972

3 Armstrong W W. Dependency Structures of Data Base Relationships. Proceedings of the IFIP Congress, 1974

(Armstrong在本文中提出了函数依赖公理系统,并给出了正确性和完备性证明。)

4 Bernstein P. Synthesizing Third Normal Form Relations from Functional Dependencies. ACM TODS, 1:4, 1976

(本文给出了关系数据库设计的综合算法(达到第三范式的综合算法)。)

5 Zaniolo C. Analysis and Design of Relational Schemata for Database Systems. Ph. D. Thesis, University of California, Los Angeles, 1976

(Zaniolo在他的博士论文中提出了多值依赖的概念。)

6 Fagin R. Multivalued Dependencies and a New Normal Form for Relational Database. ACM TODS, 2:3, 1977

(Fagin在本文中讨论并定义了第四范式。)

7 Beeri C, Fagin R, Howard J H. A Complete Axiomatization for Functional and Multivalued Dependencies. in Proceedings of ACM SIGMOD, 1977

(本文研究了函数依赖和多值依赖的公理系统及其正确性和完备性。)

8 Bernstein Philip A. Goodman Nathan. What Does Boyce-Codd Normal Form Do. in Proceedings of VLDB, 1980

9 Aho A, Beeri C, Ullman J. The Theory of Joins in Relational Databases. ACM TODS, 4:3, 1979

(本文分析了无损连接性的性质。)

10 Delobel C. An Overview of the Relational Data Theory. IFIP, 1980

11 Maier D. The Theory of Relational Databases. Rockville, Md.: Computer Science Press, 1983

(本文献中Maier对关系依赖理论进行了综合讨论,给出了许多有关函数依赖的定理及其证明,介绍了求与给定函数依赖集等价的最小函数依赖集的方法等。)

12 Kent W. Consequences of Assuming a Universal Relation. ACM TODS 6, No. 4, 1981

13 Ullman J D. On Kent's Consequences of Assuming a Universal Relation. ACM TODS 8, No. 4, 1983

第七章 数据库设计

本章讨论数据库设计的方法和技术。主要讨论基于 RDBMS 的关系数据库设计问题。

人们在总结信息资源开发、管理和服务的各种手段时，认为最有效的是数据库技术。数据库的应用已越来越广泛。从小型的单项事务处理系统到大型复杂的信息系统都用先进的数据库技术来保持系统数据的整体性、完整性和共享性。目前，一个国家的数据库建设规模（指数据库的个数、种类）、数据库信息量的大小和使用频度已成为衡量这个国家信息化程度的重要标志之一。

7.1 数据库设计概述

在数据库领域内，通常把使用数据库的各类信息系统都称为数据库应用系统。例如，以数据库为基础的各种管理信息系统、办公自动化系统、地理信息系统、电子政务系统、电子商务系统等都可以称为数据库应用系统。

什么是数据库设计呢？广义地讲，是数据库及其应用系统的设计，即设计整个的数据库应用系统。狭义地讲，是设计数据库本身，即设计数据库的各级模式并建立数据库，这是数据库应用系统设计的一部分。本书的重点是讲解狭义的数据库设计。当然设计一个好的数据库与设计一个好的数据库应用系统是密不可分的。一个好的数据库结构是应用系统的基础。特别在实际的系统开发项目中两者更是密切相关、并行进行的。

下面给出**数据库设计**的一般定义。

数据库设计是指对于一个给定的应用环境，构造（设计）优化的数据库逻辑模式和物理结构，并据此建立数据库及其应用系统，使之能够有效地存储和管理数据，满足各种用户的应用需求，包括信息管理要求和数据操作要求。

信息管理要求是指在数据库中应该存储和管理哪些数据对象；数据操作要求是指对数据对象需要进行哪些操作，如查询、增、删、改、统计等操作。

数据库设计的目标是为用户和各种应用系统提供一个信息基础设施和高效率的运行环境。高效率的运行环境包括：数据库数据的存取效率、数据库存储空间的利用率、数据库系

统运行管理的效率等都是高的。

7.1.1 数据库设计的特点

大型数据库的设计和开发是一项庞大的工程，是涉及多学科的综合性技术。数据库建设是指数据库应用系统从设计、实施到运行与维护的全过程。数据库建设和一般的软件系统的设计、开发和运行与维护有许多相同之处，更有其自身的一些特点。

一、数据库建设的基本规律

“三分技术，七分管理，十二分基础数据”是数据库设计的特点之一。

在数据库建设中不仅涉及技术，还涉及管理。要建设好一个数据库应用系统，开发技术固然重要，但是相比之下则管理更加重要。这里的管理不仅仅包括数据库建设作为一个大型的工程项目本身的项目管理，而且包括该企业（即应用部门）的业务管理。

企业的业务管理更加复杂，也更重要，对数据库结构的设计有直接影响。这是因为数据库结构（即数据库模式）是对企业中业务部门的数据以及各个业务部门之间数据联系的描述和抽象。业务部门数据以及各个业务部门之间数据的联系是和各个部门的职能、整个企业的管理模式密切相关的。

人们在数据库建设的长期实践中深刻认识到一个企业数据库建设的过程是企业管理模式的改革和提高的过程。只有把企业的管理创新做好，才能实现技术创新，才能建设好一个数据库应用系统。

十二分基础数据则强调了数据的收集、整理、组织和不断更新是数据库建设中的重要环节。人们往往忽视基础数据在数据库建设中的地位和作用。

基础数据的收集、入库是数据库建立初期工作量最大、最烦琐、最细致的工作。在以后数据库运行过程中更需要不断地把新的数据加到数据库中，使数据库成为一个“活库”，否则就成了“死库”。数据库一旦成了“死库”，系统也就失去了应用价值，原来的投资也就失败了。

二、结构（数据）设计和行为（处理）设计相结合

数据库设计应该和应用系统设计相结合。也就是说，整个设计过程中要把数据库结构设计和对数据的处理设计密切结合起来。这是数据库设计的特点之二。

但是在早期的数据库应用系统开发过程中，常常把数据库设计和应用系统的设计分离开来，如图 7.1 所示。由于数据库设计有它专门的技术和理论，因此需要专门来讲解数据库设计。这并不等于数据库设计和在数据库之上开发应用系统是相互分离的。相反，必须强调设计过程中数据库设计和应用程序设计的密切结合，并把它作为数据库设计的重要特点。

传统的软件工程忽视对应用中数据语义的分析和抽象。例如结构化设计（Structure Design，SD）方法和逐步求精的方法着重于处理过程的特性，只要有可能就尽量推迟数据结构设计的决策。这种方法对于数据库应用系统的设计显然是不妥的。

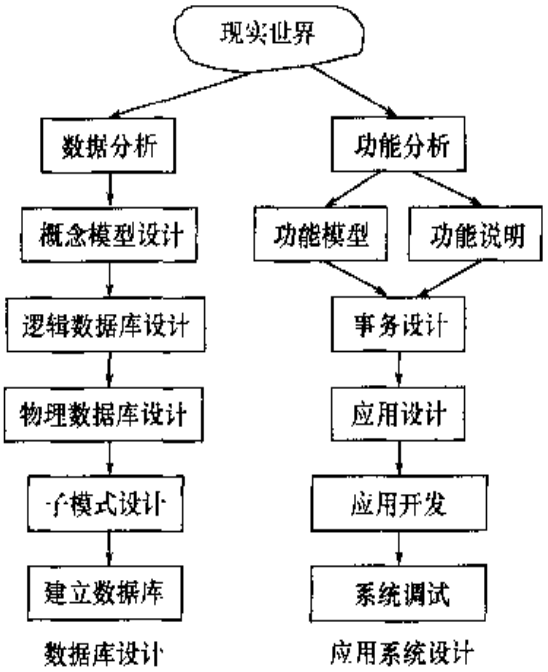


图 7.1 结构和行为分离的设计

早期的数据库设计致力于数据模型和数据库建模方法的研究，着重结构特性的设计而忽视了行为的设计对结构设计的影响，这种方法也是不完善的。

我们则强调在数据库设计中要把结构特性和行为特性结合起来。

7.1.2 数据库设计方法

大型数据库设计是涉及多学科的综合性技术，又是一项庞大的工程项目。它要求从事数据库设计的专业人员具备多方面的技术和知识。主要包括：

- 计算机的基础知识；
- 软件工程的原理和方法；
- 程序设计的方法和技巧；
- 数据库的基本知识；
- 数据库设计技术；
- 应用领域的知识。

这样才能设计出符合具体领域要求的数据库及其应用系统。

早期数据库设计主要采用手工与经验相结合的方法。设计的质量往往与设计人员的经验与水平有直接的关系。数据库设计是一种技艺，缺乏科学理论和工程方法的支持，设计质量难以保证。常常是数据库运行一段时间后又不同程度地发现各种问题，需要进行修改甚至重新设计，增加了系统维护的代价。

为此，人们努力探索，提出了各种数据库设计方法。例如：

• 新奥尔良(New Orleans)方法。该方法把数据库设计分为若干阶段和步骤,并采用一些辅助手段实现每一过程。它运用软件工程的思想,按一定的设计规程用工程化方法设计数据库。新奥尔良方法属于规范设计法。

规范设计法从本质上看仍然是手工设计方法,其基本思想是过程迭代和逐步求精。

• 基于E-R模型的数据库设计方法。该方法用E-R模型来设计数据库的概念模型,是数据库概念设计阶段广泛采用的方法。

• 3NF(第三范式)的设计方法。该方法用关系数据理论为指导来设计数据库的逻辑模型,是设计关系数据库时在逻辑阶段可以采用的一种有效方法。

• ODL(Object Definition Language)方法。这是面向对象的数据库设计方法。该方法用面向对象的概念和术语来说明数据库结构。ODL可以描述面向对象数据库结构设计,可以直接转换为面向对象的数据库。

数据库工作者一直在研究和开发数据库设计工具。经过多年的努力,数据库设计工具已经实用化和产品化。例如,Designer2000和PowerDesigner分别是ORACLE公司和SYBASE公司推出的数据库设计工具软件,这些工具软件可以辅助设计人员完成数据库设计过程中的很多任务,已经普遍地用于大型数据库设计之中。

7.1.3 数据库设计的基本步骤

按照规范设计的方法,考虑数据库及其应用系统开发全过程,将数据库设计分为以下6个阶段(如图7.2所示):

- 需求分析;
- 概念结构设计;
- 逻辑结构设计;
- 物理结构设计;
- 数据库实施;
- 数据库运行和维护。

在数据库设计过程中,需求分析和概念设计可以独立于任何数据库管理系统进行。逻辑设计和物理设计与选用的DBMS密切相关。

数据库设计开始之前,首先必须选定参加设计的人员,包括系统分析人员、数据库设计人员、应用开发人员、数据库管理员和用户代表。系统分析和数据库设计人员是数据库设计的核心人员,他们将自始至终参与数据库设计,他们的水平决定了数据库系统的质量。用户和数据库管理员在数据库设计中也是举足轻重的,他们主要参加需求分析和数据库的运行和维护,他们的积极参与(不仅仅是配合)不但能加速数据库设计,而且也是决定数据库设计质量的重要因素。应用开发人员(包括程序员和操作员)分别负责编制程序和准备软硬件环境,他们在系统实施阶段参与进来。

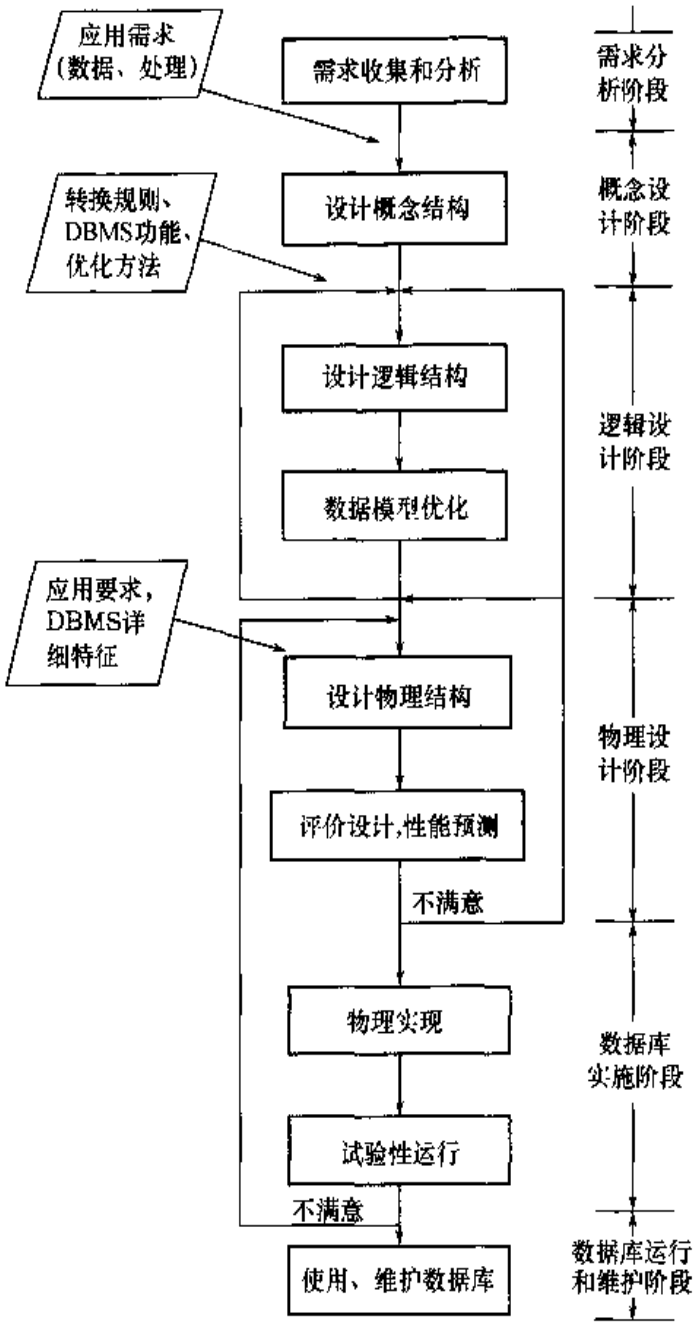


图 7.2 数据库设计步骤

如果所设计的数据库应用系统比较复杂,还应该考虑是否需要使用数据库设计工具以及选用何种工具,以提高数据库设计质量并减少设计工作量。

1. 需求分析阶段

进行数据库设计首先必须准确了解与分析用户需求(包括数据与处理)。需求分析是整个设计过程的基础,是最困难、最耗费时间的一步。作为"地基"的需求分析是否做得充分与准确,决定了在其上构建数据库大厦的速度与质量。需求分析做得不好,甚至会导致整个数据库设计返工重做。

2. 概念结构设计阶段

概念结构设计是整个数据库设计的关键，它通过对用户需求进行综合、归纳与抽象，形成一个独立于具体 DBMS 的概念模型。

3. 逻辑结构设计阶段

逻辑结构设计是将概念结构转换为某个 DBMS 所支持的数据模型，并对其进行优化。

4. 物理设计阶段

物理设计是为逻辑数据模型选取一个最适合应用环境的物理结构（包括存储结构和存取方法）。

5. 数据库实施阶段

在数据库实施阶段，设计人员运用 DBMS 提供的数据库语言（如 SQL）及其宿主语言，根据逻辑设计和物理设计的结果建立数据库，编制与调试应用程序，组织数据入库，并进行试运行。

6. 数据库运行和维护阶段

数据库应用系统经过试运行后即可投入正式运行。在数据库系统运行过程中必须不断地对其进行评价、调整与修改。

设计一个完善的数据库应用系统是不可能一蹴而就的，它往往是上述 6 个阶段的不断反复。

需要指出的是，这个设计步骤既是数据库设计的过程，也包括了数据库应用系统的设计过程。在设计过程中把数据库的设计和对数据库中数据处理的设计紧密结合起来，将这两个方面的需求分析、抽象、设计、实现在各个阶段同时进行，相互参照，相互补充，以完善两方面的设计。事实上，如果不了解应用环境对数据的处理要求，或没有考虑如何去实现这些处理要求，是不可能设计一个良好的数据库结构的。按照这个原则，设计过程各个阶段的设计描述，可用图 7.3 概括地给出。

图 7.3 的有关处理特性的设计描述中，其设计原理、采用的设计方法、工具等在软件工程和信息系统设计的课程中有详细介绍，这里不再讨论。这里主要讨论关于数据特性的描述以及如何在整个设计过程中参照处理特性的设计来完善数据模型设计等问题。

7.1.4 数据库设计过程中的各级模式

按照 7.1.3 小节的设计过程，数据库设计的不同阶段形成数据库的各级模式，如图 7.4 所示。需求分析阶段，综合各个用户的应用需求；在概念设计阶段形成独立于机器特点，独立于各个 DBMS 产品的概念模式，在本篇中就是 E-R 图；在逻辑设计阶段将 E-R 图转换成具体的数据库产品支持的数据模型，如关系模型，形成数据库逻辑模式；然后根据用户处理的要求、安全性的考虑，在基本表的基础上再建立必要的视图（View），形成数据的外模式；在物理设计阶段，根据 DBMS 特点和处理的需要，进行物理存储安排，建立索引，形成数据库内

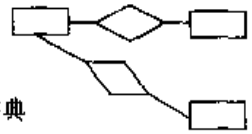
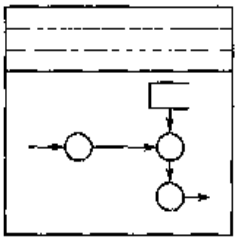
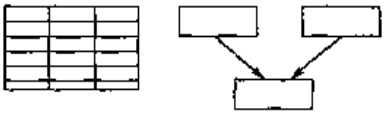
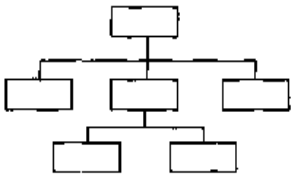
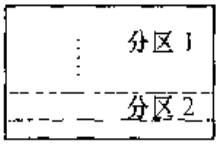
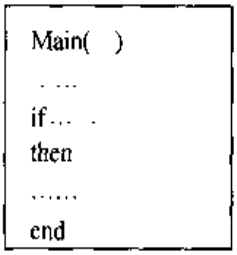
设计阶段	设计描述	
	数据	处理
需求分析	数据字典、全系统中数据项、数据流、数据存储的描述	数据流图和判定表（判定树）、数据字典中处理过程的描述
概念结构设计	概念模型（E-R图）  数据字典	系统说明书包括： ① 新系统要求、方案和概图 ② 反映新系统信息流的数据流图 
逻辑结构设计	某种数据模型 关系 非关系 	系统结构图 （模块结构） 
物理设计	存储安排 方法选择 存取路径建立 	模块设计 IPO表 
数据库实施阶段	编写模式 装入数据 数据库试运行 	程序编码、编译联结、测试 
数据库运行和维护	性能监测、转储/恢复 数据库重组和重构	新旧系统转换、运行、维护（修正性、适应性、改善性维护）

图 7.3 数据库设计各个阶段的设计描述

模式。

下面就以图 7.2 的设计过程为主线，讨论数据库设计各个阶段的设计内容、设计方法和工具。

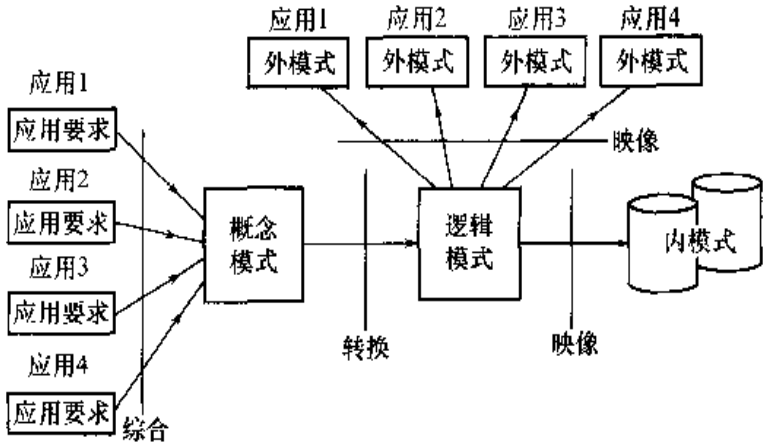


图 7.4 数据库的各级模式

7.2 需求分析

需求分析简单地说就是分析用户的要求。需求分析是设计数据库的起点，需求分析的结果是否准确地反映了用户的实际要求，将直接影响到后面各个阶段的设计，并影响到设计结果是否合理和实用。

7.2.1 需求分析的任务

需求分析的任务是通过详细调查现实世界要处理的对象（组织、部门、企业等），充分了解原系统（手工系统或计算机系统）工作概况，明确用户的各种需求，然后在此基础上确定新系统的功能。新系统必须充分考虑今后可能的扩充和改变，不能仅仅按当前应用需求来设计数据库。

调查的重点是“数据”和“处理”，通过调查、收集与分析，获得用户对数据库如下要求：

（1）信息要求。指用户需要从数据库中获得信息的内容与性质。由信息要求可以导出数据要求，即在数据库中需要存储哪些数据。

（2）处理要求。指用户要完成什么处理功能，对处理的响应时间有什么要求，处理方式是批处理还是联机处理。

（3）安全性与完整性要求。

确定用户的最终需求是一件很困难的事，这是因为一方面用户缺少计算机知识，开始时无法确定计算机究竟能为自己做什么，不能做什么，因此往往不能准确地表达自己的需求，所提出的需求往往不断地变化。另一方面，设计人员缺少用户的专业知识，不易理解用户的真正需求，甚至误解用户的需求。因此设计人员必须不断深入地与用户交流，才能逐步确定用户的实际需求。

7.2.2 需求分析的方法

进行需求分析首先是调查清楚用户的实际要求，与用户达成共识，然后分析与表达这些需求。

调查用户需求的具体步骤是：

(1) 调查组织机构情况。包括了解该组织的部门组成情况、各部门的职责等，为分析信息流程做准备。

(2) 调查各部门的业务活动情况。包括了解各个部门输入和使用什么数据，如何加工处理这些数据，输出什么信息，输出到什么部门，输出结果的格式是什么，这是调查的重点。

(3) 在熟悉了业务活动的基础上，协助用户明确对新系统的各种要求，包括信息要求、处理要求、安全性与完整性要求，这是调查的又一个重点。

(4) 确定新系统的边界。对前面调查的结果进行初步分析，确定哪些功能由计算机完成或将来准备让计算机完成，哪些活动由人工完成。由计算机完成的功能就是新系统应该实现的功能。

在调查过程中，可以根据不同的问题和条件，使用不同的调查方法。常用的调查方法有：

(1) 跟班作业。通过亲身参加业务工作来了解业务活动的情况。

(2) 开调查会。通过与用户座谈来了解业务活动情况及用户需求。

(3) 请专人介绍。

(4) 询问。对某些调查中的问题，可以找专人询问。

(5) 设计调查表请用户填写。如果调查表设计得合理，这种方法是很有效的。

(6) 查阅记录。查阅与原系统有关的数据记录。

做需求调查时，往往需要同时采用上述多种方法。但无论使用何种调查方法，都必须有用户的积极参与和配合。

调查了解了用户需求以后，还需要进一步分析和表达用户的需求。在众多的分析方法中结构化分析(Structured Analysis，SA)方法是一种简单实用的方法。SA 方法从最上层的系统组织机构入手，采用自顶向下、逐层分解的方式分析系统。SA 方法把任何一个系统都抽象为图 7.5 所示的形式。

图 7.5 给出的只是最高层次抽象的系统概貌，要反映更详细的内容，可将处理功能分解为若干子功能，每个子功能还可以继续分解，直到把系统工作过程表示清楚为止。在处理功能逐步分解的同时，它们所用的数据也逐级分解，形成若干层次的数据流图。

数据流图表达了数据和处理过程的关系。在 SA 方法中，处理过程的处理逻辑常常借助判定表或判定树来描述。系统中的数据则借助数据字典(Data Dictionary，DD)来描述。

对用户需求进行分析与表达后，必须提交给用户，征得用户的认可。

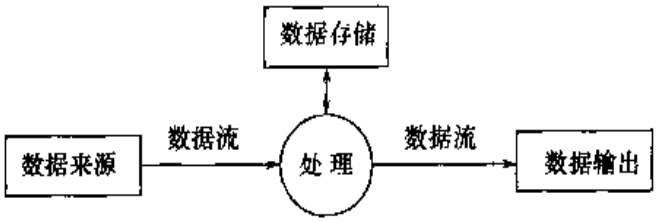


图 7.5 系统高层抽象图

图 7.6 描述了需求分析的过程。

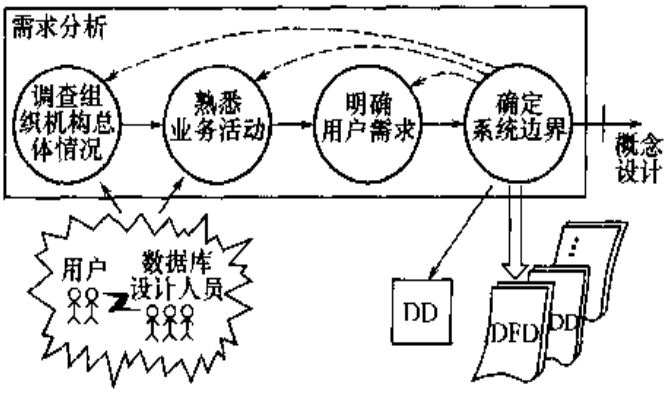


图 7.6 需求分析过程

7.2.3 数据字典

数据流图表达了数据和处理的关系，数据字典则是系统中各类数据描述的集合，是进行详细的数据收集和数据分析所获得的主要成果。数据字典在数据库设计中占有很重要的地位。

数据字典通常包括数据项、数据结构、数据流、数据存储和处理过程 5 个部分。其中数据项是数据的最小组成单位，若干个数据项可以组成一个数据结构，数据字典通过对数据项和数据结构的定义来描述数据流、数据存储的逻辑内容。

1. 数据项

数据项是不可再分的数据单位。对数据项的描述通常包括以下内容：

数据项描述 = {数据项名，数据项含义说明，别名，数据类型，长度，取值范围，
取值含义，与其他数据项的逻辑关系，数据项之间的联系}

其中，“取值范围”、“与其他数据项的逻辑关系”（例如，该数据项等于另几个数据项的和，该数据项值等于另一数据项的值等）定义了数据的完整性约束条件，是设计数据检验功能的依据。

可以用关系规范化理论为指导，用数据依赖的概念分析和表示数据项之间的联系。即按实际语义，写出每个数据项之间的数据依赖，它们是数据库逻辑设计阶段数据模型优化的

依据。

2. 数据结构

数据结构反映了数据之间的组合关系。一个数据结构可以由若干个数据项组成，也可以由若干个数据结构组成，或由若干个数据项和数据结构混合组成。对数据结构的描述通常包括以下内容：

数据结构描述 = {数据结构名，含义说明，组成：{数据项或数据结构}}

3. 数据流

数据流是数据结构在系统内传输的路径。对数据流的描述通常包括以下内容：

数据流描述 = {数据流名，说明，数据流来源，数据流去向，
组成：{数据结构}，平均流量，高峰期流量}

其中，“数据流来源”是说明该数据流来自哪个过程；“数据流去向”是说明该数据流将到哪个过程去；“平均流量”是指在单位时间（每天、每周、每月等）里的传输次数；“高峰期流量”则是指在高峰时期的数据流量。

4. 数据存储

数据存储是数据结构停留或保存的地方，也是数据流的来源和去向之一。它可以是手工文档或手工凭单，也可以是计算机文档。对数据存储的描述通常包括以下内容：

数据存储描述 = {数据存储名，说明，编号，输入的数据流，输出的数据流，
组成：{数据结构}，数据量，存取频度，存取方式}

其中，“存取频度”指每小时或每天或每周存取几次、每次存取多少数据等信息；“存取方式”包括是批处理还是联机处理、是检索还是更新、是顺序检索还是随机检索等；另外，“输入的数据流”要指出其来源；“输出的数据流”要指出其去向。

5. 处理过程

处理过程的具体处理逻辑一般用判定表或判定树来描述。数据字典中只需要描述处理过程的说明性信息，通常包括以下内容：

处理过程描述 = {处理过程名，说明，输入：{数据流}，输出：{数据流}，
处理：{简要说明}}

其中，“简要说明”中主要说明该处理过程的功能及处理要求。功能是指该处理过程用来做什么（而不是怎么做），处理要求包括处理频度要求，如单位时间里处理多少事务、多少数据量、响应时间要求等。这些处理要求是后面物理设计的输入及性能评价的标准。

可见，数据字典是关于数据库中数据的描述，即元数据，而不是数据本身。

数据字典是在需求分析阶段建立，在数据库设计过程中不断修改、充实、完善的。

明确地把需求收集和分析作为数据库设计的第一阶段是十分重要的。这一阶段收集到的基础数据（用数据字典来表达）和一组数据流程图（Data Flow Diagram，DFD）是下一步进行概念设计的基础。

最后，要强调两点：

(1) 需求分析阶段的一个重要而困难的任务是收集将来应用所涉及的数据,设计人员应充分考虑到可能的扩充和改变,使设计易于更改,系统易于扩充,这是第一点。

(2) 必须强调用户的参与,这是数据库应用系统设计的特点。数据库应用系统和广泛的用户有密切的联系,许多人要使用数据库。数据库的设计和建立又可能对更多人的工作环境产生重要影响。因此用户的参与是数据库设计不可分割的一部分。在数据分析阶段,任何调查研究没有用户的积极参与是寸步难行的。设计人员应该和用户取得共同的语言,帮助不熟悉计算机的用户建立数据库环境下的共同概念,并对设计工作的最后结果承担共同的责任。

7.3 概念结构设计

将需求分析得到的用户需求抽象为信息结构即概念模型的过程就是概念结构设计。它是整个数据库设计的关键。

7.3.1 概念结构

在需求分析阶段所得到的应用需求应该首先抽象为信息世界的结构,才能更好地、更准确地用某一 DBMS 实现这些需求。

概念结构的主要特点是:

(1) 能真实、充分地反映现实世界,包括事物和事物之间的联系,能满足用户对数据的处理要求,是对现实世界的一个真实模型;

(2) 易于理解,从而可以用它和不熟悉计算机的用户交换意见,用户的积极参与是数据库设计成功的关键;

(3) 易于更改,当应用环境和应用要求改变时,容易对概念模型修改和扩充;

(4) 易于向关系、网状、层次等各种数据模型转换。

概念结构是各种数据模型的共同基础,它比数据模型更独立于机器、更抽象,从而更加稳定。

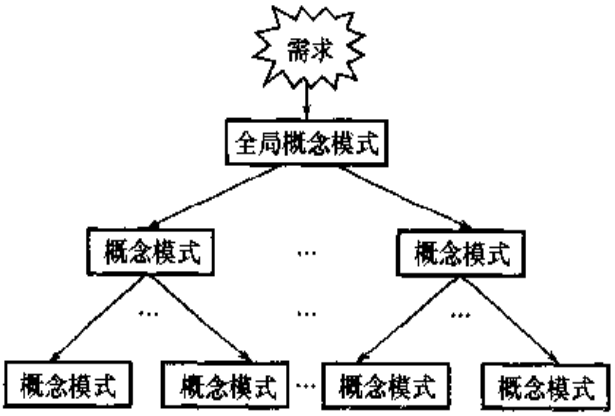
描述概念模型的有力工具是 E-R 模型。有关 E-R 模型的基本概念已在第一章介绍。下面将用 E-R 模型来描述概念结构。

7.3.2 概念结构设计的方法与步骤

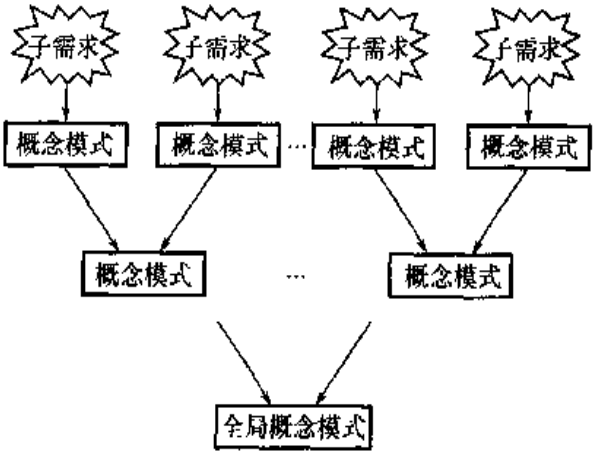
设计概念结构通常有四类方法:

- 自顶向下。即首先定义全局概念结构的框架,然后逐步细化,如图 7.7(a)所示。
- 自底向上。即首先定义各局部应用的概念结构,然后将它们集成起来,得到全局概念结构,如图 7.7(b)所示。

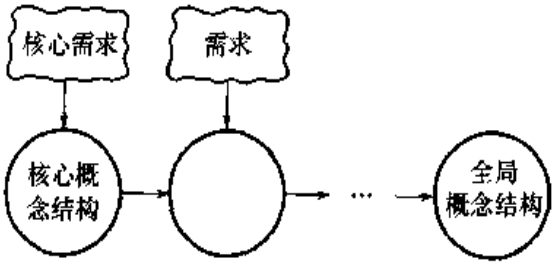
• 逐步扩张。首先定义最重要的核心概念结构，然后向外扩充，以滚雪球的方式逐步生成其他概念结构，直至总体概念结构，如图 7.7(c)所示。



(a) 自顶向下策略



(b) 自底向上策略



(c) 逐步扩张策略

图 7.7　设计概念结构的策略

• 混合策略。即将自顶向下和自底向上相结合，用自顶向下策略设计一个全局概念结构的框架，以它为骨架集成由自底向上策略中设计的各局部概念结构。

其中最经常采用的策略是自底向上方法，即自顶向下地进行需求分析，然后再自底向上地设计概念结构，如图 7.8 所示。

这里只介绍自底向上设计概念结构的方法。它通常分为两步：第一步是抽象数据并设计局部视图，第二步是集成局部视图，得到全局的概念结构，如图 7.9 所示。

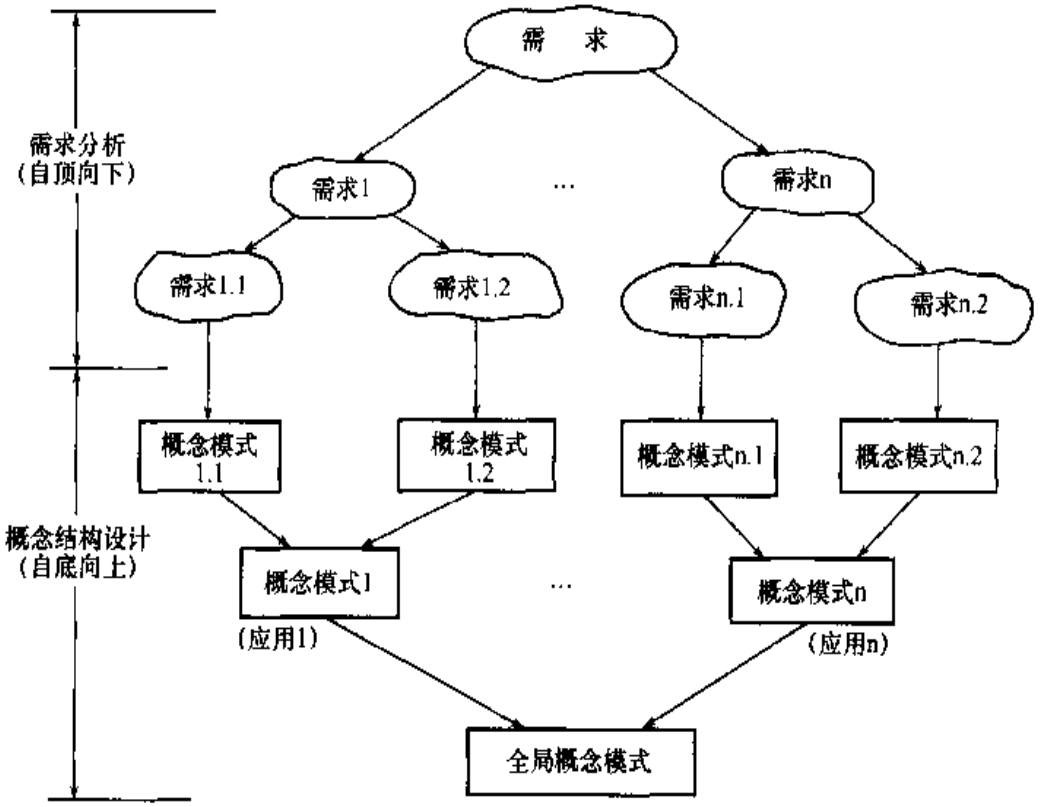


图 7.8 自底向上设计概念结构

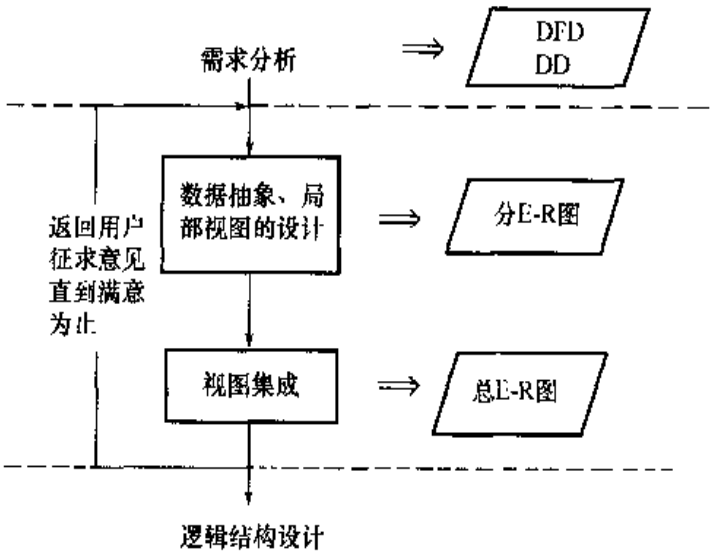


图 7.9 概念结构设计

7.3.3　数据抽象与局部视图设计

概念结构是对现实世界的一种抽象。所谓抽象是对实际的人、物、事和概念进行人为处理，抽取所关心的共同特性，忽略非本质的细节，并把这些特性用各种概念精确地加以描述，这些概念组成了某种模型。

一般有三种抽象：

1. 分类(Classification)

定义某一类概念作为现实世界中一组对象的类型。这些对象具有某些共同的特性和行为。它抽象了对象值和型之间的"is member of"的语义。在E-R模型中，实体型就是这种抽象。例如，在学校环境中，张英是学生(如图7.10所示)，表示张英是学生中的一员，具有学生们共同的特性和行为：在某个班学习某种专业，选修某些课程。

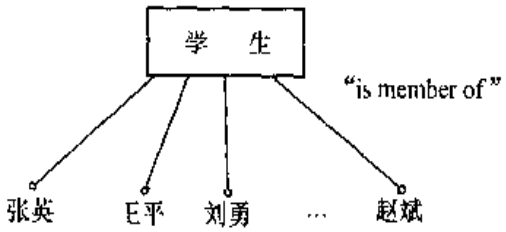


图7.10　分类

2. 聚集(Aggregation)

定义某一类型的组成成分。它抽象了对象内部类型和成分之间"is part of"的语义。在E-R模型中若干属性的聚集组成了实体型，就是这种抽象，如图7.11所示。

更复杂的聚集如图7.12所示，即某一类型的成分仍是一个聚集。

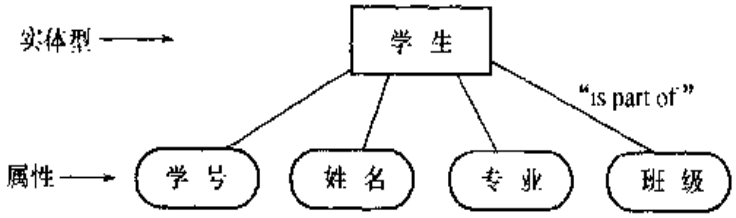


图7.11　聚集

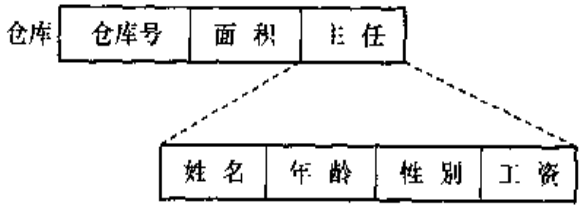


图7.12　更复杂的聚集

3. 概括(Generalization)

定义类型之间的一种子集联系。它抽象了类型之间的"is subset of"的语义。例如，学生是一个实体型，本科生、研究生也是实体型。本科生、研究生均是学生的子集。把学生称为超类(Superclass)，本科生、研究生称为学生的子类(Subclass)。

原E-R模型不具有概括，本书对E-R模型作了扩充，允许定义超类实体型和子类实体

型。并用双竖边的矩形框表示子类,用直线加小圆圈表示超类 - 子类的联系(如图 7.13 所示)。

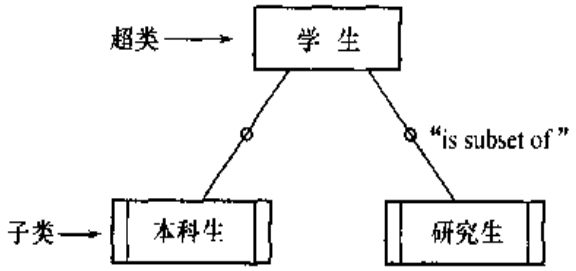


图 7.13 概括

概括有一个很重要的性质:继承性。子类继承超类上定义的所有抽象。这样,本科生、研究生继承了学生类型的属性。当然,子类可以增加自己的某些特殊属性。

概念结构设计的第一步就是利用上面介绍的抽象机制对需求分析阶段收集到的数据进行分类、组织(聚集),形成实体、实体的属性,标识实体的码,确定实体之间的联系类型(1:1,1:n,m:n),设计分 E-R 图。具体做法是:

1. 选择局部应用

根据某个系统的具体情况,在多层的数据流图中选择一个适当层次的数据流图,作为设计分 E-R 图的出发点。让这组图中每一部分对应一个局部应用。

由于高层的数据流图只能反映系统的概貌,而中层的数据流图能较好地反映系统中各局部应用的子系统组成,因此人们往往以中层数据流图作为设计分 E-R 图的依据(如图 7.14 所示)。

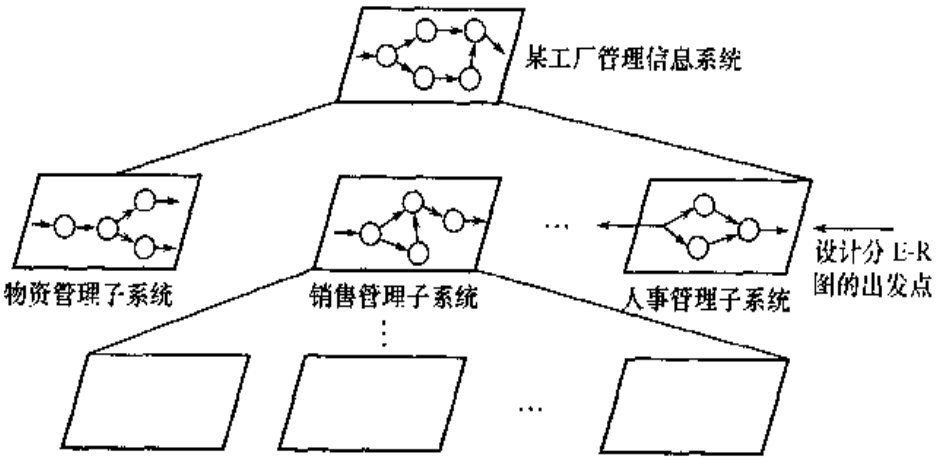


图 7.14 设计分 E-R 图的出发点

2. 逐一设计分 E-R 图

选择好局部应用之后,就要对每个局部应用逐一设计分 E-R 图。

在前面选好的某一层次的数据流图中,每个局部应用都对应了一组数据流图,局部应用涉及的数据都已经收集在数据字典中了。现在就是要将这些数据从数据字典中抽取出来,参照数据流图,标定局部应用中的实体、实体的属性、标识实体的码,确定实体之间的联系及其类型。

事实上,在现实世界中具体的应用环境常常对实体和属性已经作了大体的自然的划分。

在数据字典中，“数据结构”、“数据流”和“数据存储”都是若干属性有意义的聚合，就体现了这种划分。可以先从这些内容出发定义 E-R 图，然后再进行必要的调整。在调整中遵循的一条原则是：

为了简化 E-R 图的处置，现实世界的事物能作为属性对待的，尽量作为属性对待。

那么符合什么条件的事物可以作为属性对待呢？本来，实体与属性之间并没有形式上可以截然划分的界限，但可以给出两条准则：

(1) 作为“属性”，不能再具有需要描述的性质。“属性”必须是不可分的数据项，不能包含其他属性。

(2) “属性”不能与其他实体具有联系，即 E-R 图中所表示的联系是实体之间的联系。

凡满足上述两条准则的事物，一般均可作为属性对待。

例如，职工是一个实体，职工号、姓名、年龄是职工的属性，职称如果没有与工资、福利挂钩，换句话说，没有需要进一步描述的特性，则根据准则(1)可以作为职工实体的属性；但如果不同的职称有不同的工资、住房标准和不同的附加福利，则职称作为一个实体看待就更恰当，如图 7.15 所示。

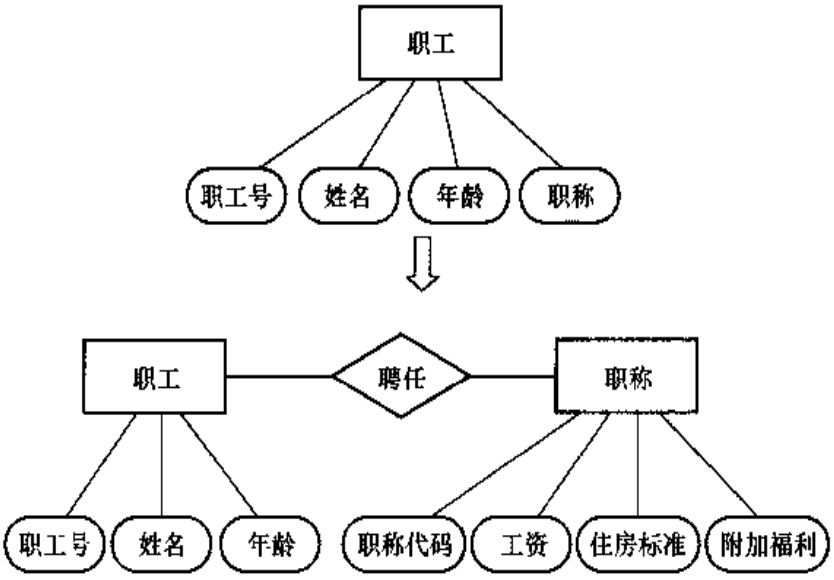


图 7.15　职称作为一个实体

再如，在医院中，一个病人只能住在一个病房，病房号可以作为病人实体的一个属性；但如果病房还要与医生实体发生联系，即一个医生负责几个病房的病人的医疗工作，则病房根据准则(2)应作为一个实体，如图 7.16 所示。

又如，如果一种货物只存放在一个仓库，那么就可以把存放货物的仓库的仓库号作为描述货物存放地点的属性，但如果一种货物可以存放在多个仓库中，或者仓库本身又用面积作为属性，或者与职工发生管理上的联系，那么就应把仓库作为一个实体，如图 7.17 所示。

[**实例**]　销售管理子系统分 E-R 图的设计。

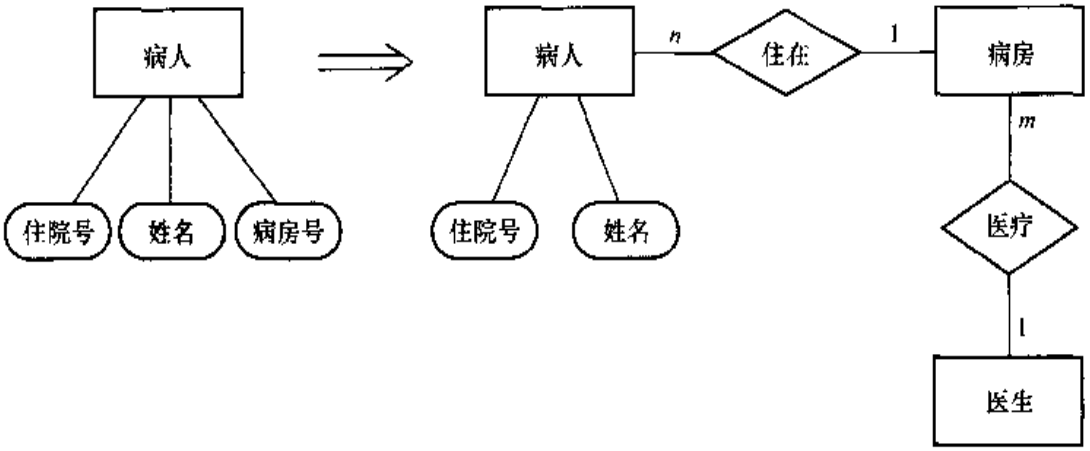


图 7.16 病房作为一个实体

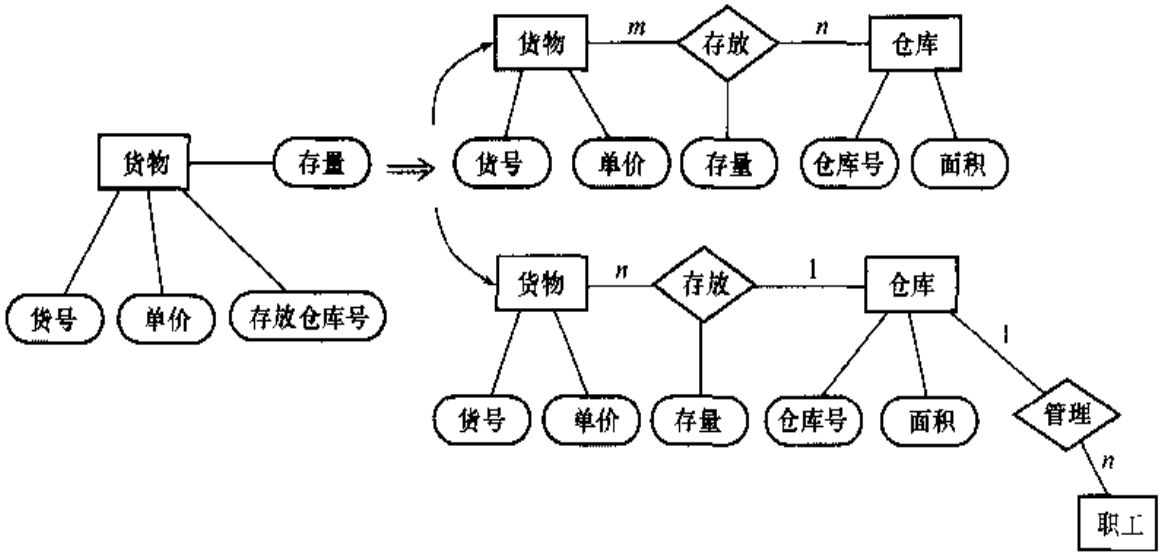


图 7.17 仓库作为一个实体

某工厂开发管理信息系统,经过可行性分析,详细调查确定了该系统由物资管理、销售管理、劳动人事管理等子系统组成。为每个子系统组成了开发小组。

销售管理子系统开发小组的成员经过调查研究、信息流程分析和数据收集,明确了该子系统的主要功能是:处理顾客和销售员送来的订单;工厂是根据订货安排生产的;交出货物同时开出发票;收到顾客付款后,根据发票存根和信贷情况进行应收款处理。得到了该子系统二层数据流图(共 5 张)和数据字典,其中包括 14 个数据结构和 29 个数据流(见本章参考文献[6])。

图 7.18 是第一层数据流图。虚线部分划出了系统边界。图中把系统功能又分为 4 个子系统。图 7.19 ~ 7.22 是第二层数据流图。

由于该子系统不太复杂,设计分 E-R 图可以从图 7.18 第一层数据流图入手。若某一局部应用仍比较复杂,则可以从更下层的数据流图入手。例如,从图 7.19 ~ 7.22 开始,分别设

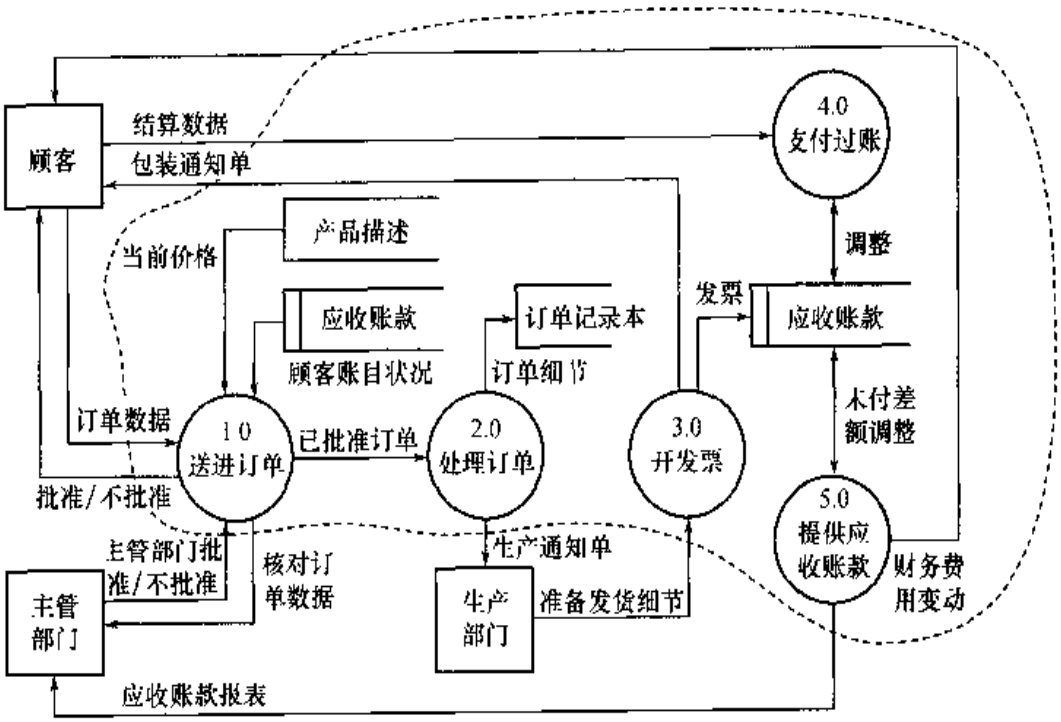


图 7.18 销售管理子系统第一层数据流图

计它们的分 E-R 图,再汇总成该局部应用的分 E-R 图。

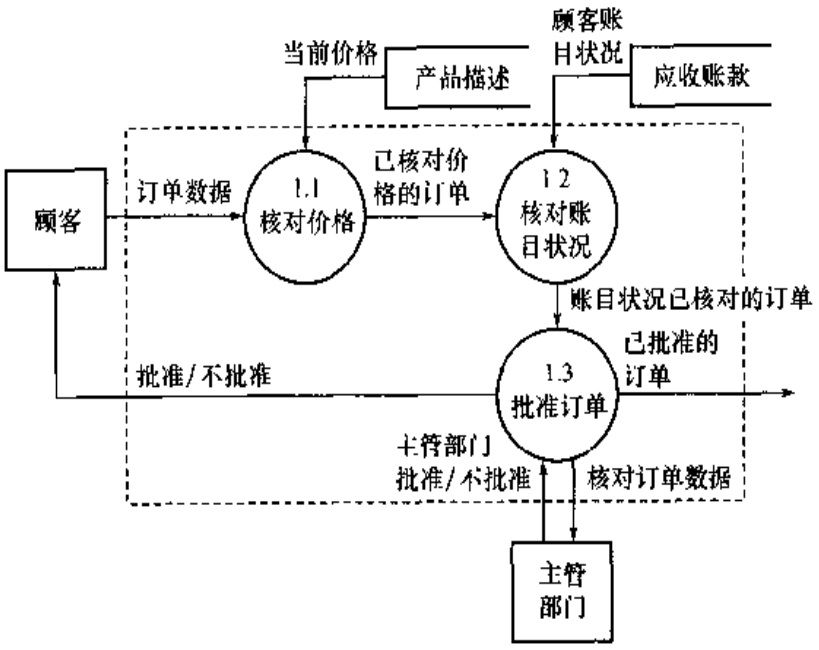


图 7.19 接收订单

分析图 7.18 和数据字典,知道整个系统功能围绕了“订单”和“应收账款”的处理。数据结构中订单、顾客、顾客应收账目用得最多,是许多子功能、数据流共享的数据,因此先设计该分 E-R 图的草图(如图 7.23 所示)。

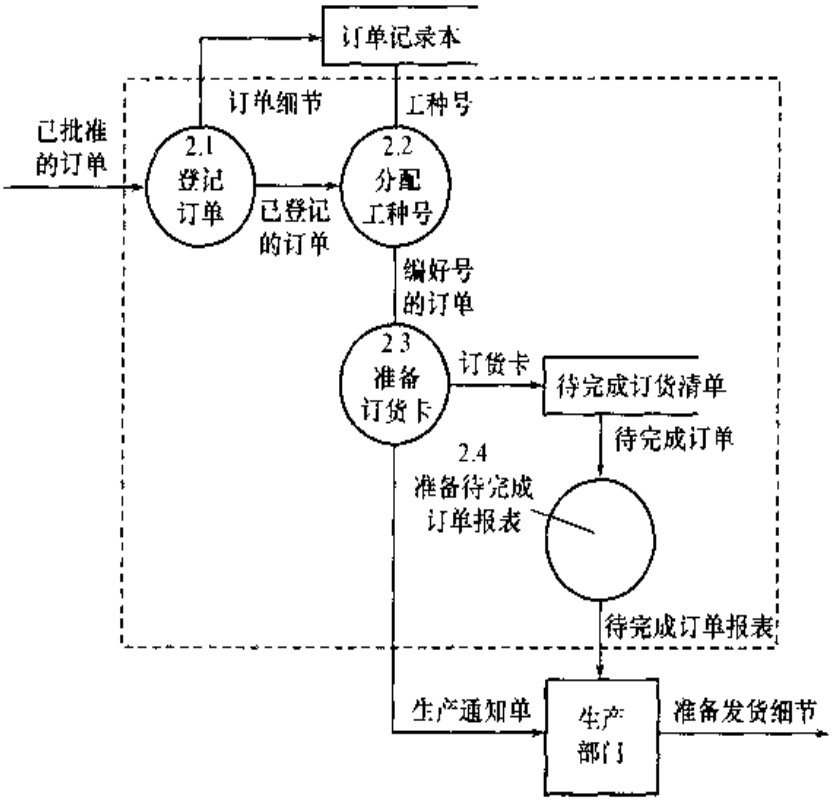


图 7.20 处理订单

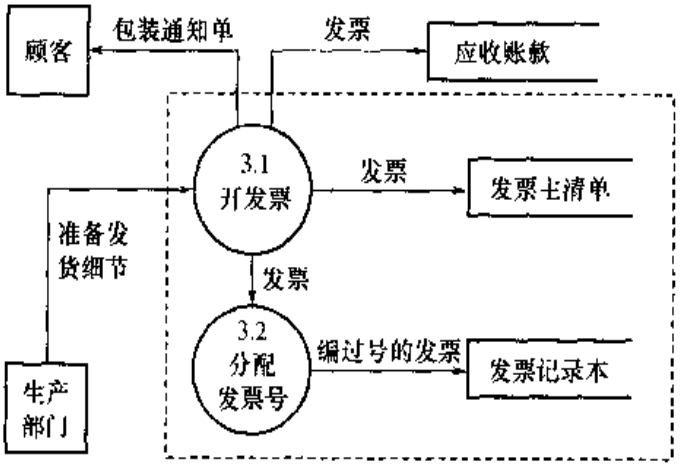


图 7.21 开发票

然后参照第二层数据流图和数据字典中的详尽描述,遵循前面给出的两个准则,进行了如下调整:

(1) 每张订单由订单号、若干头信息和订单细节组成。订单细节又有订货的零件号、数量等来描述。按照准则(2),订单细节就不能作订单的属性处理而应该上升为实体。一张订单可以订若干产品,所以订单与订单细节两个实体之间是 1:n 的联系。

(2) 原订单和产品的联系实际上是订单细节和产品的联系。每条订货细节对应一个产

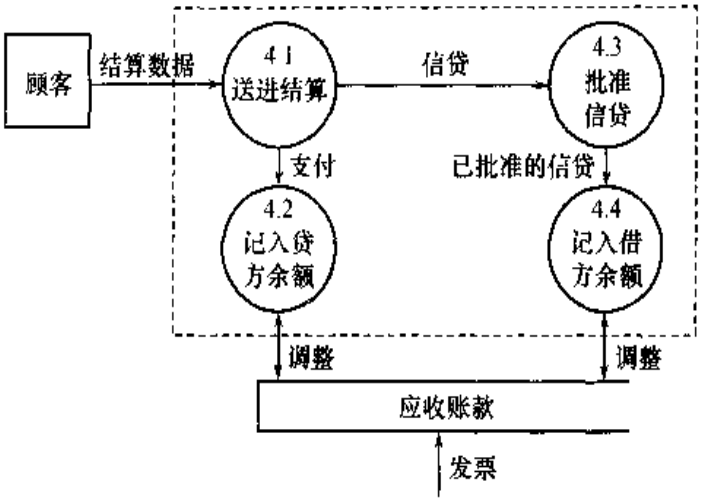


图 7.22 支付过账

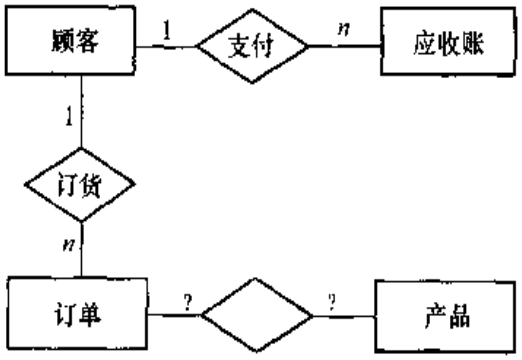


图 7.23 分 E-R 图的框架

品描述,订单处理时从中获得当前单价、产品重量等信息。

(3) 图 7.21 中"发票主清单"是一个数据存储,是否应作为实体加入分 E-R 图呢? 答案是不必。这里的数据存储对应手工凭证,发票上的信息在开具发票同时已及时存入应收账款中了。

(4) 工厂对大宗订货给予优惠。每种产品都规定了不同订货数量的折扣,应增加一个"折扣规则"实体存放这些信息,而不应把它们放在产品描述实体中。

最后得到分 E-R 图如图 7.24 所示。

对每个实体定义的属性如下:

顾客:{顾客号,顾客名,地址,电话,信贷状况,账目余额}
订单:{订单号,顾客号,订货项数,订货日期,交货日期,工种号,生产地点}
订单细则:{订单号,细则号,零件号,订货数,金额}
应收账款:{顾客号,订单号,发票号,应收金额,支付日期,支付金额,
当前余额,货款限额}
产品描述:{产品号,产品名,单价,重量}

折扣规则:{产品号,订货量,折扣}

(**注意**:为了节省篇幅,实体与属性的关系没有用图形表示,实体的标识码用下横线划出)。

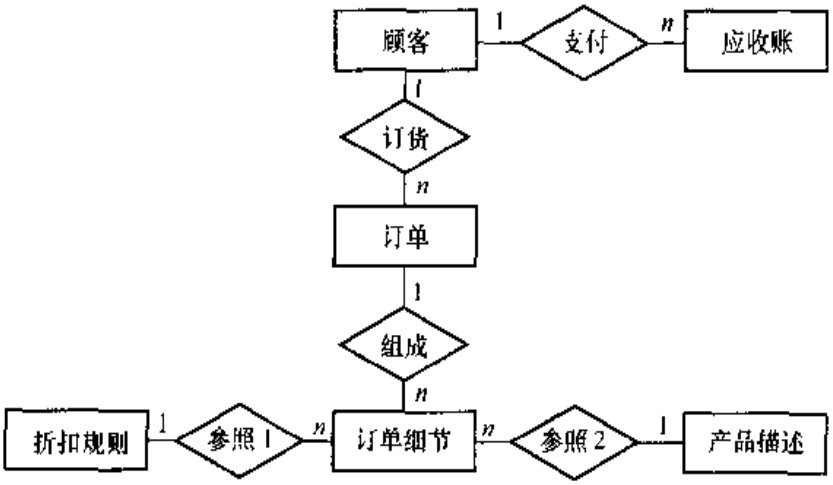


图 7.24 销售管理子系统的分 E-R 图

7.3.4 视图的集成

各子系统的分 E-R 图设计好以后,下一步就是要将所有的分 E-R 图综合成一个系统的总 E-R 图。一般说来,视图集成可以有两种方式:

- 多个分 E-R 图一次集成,如图 7.25(a)所示。
- 逐步集成,用累加的方式一次集成两个分 E-R 图,如图 7.25(b)所示。

第一种方法比较复杂,做起来难度较大。

第二种方法每次只集成两个分 E-R 图,可以降低复杂度。

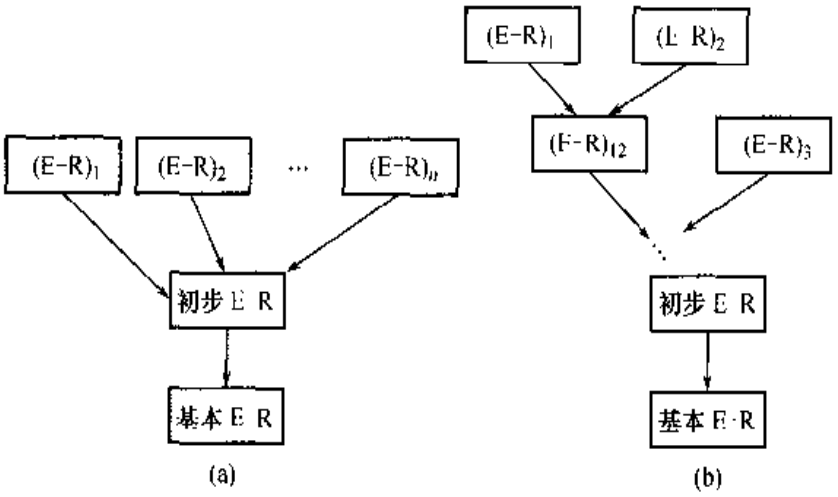


图 7.25 视图集成的两种方式

无论采用哪种方式，每次集成局部 E-R 图时都需要分两步走，如图 7.26 所示。

(1) 合并。解决各分 E-R 图之间的冲突，将各分 E-R 图合并起来生成初步 E-R 图。

(2) 修改和重构。消除不必要的冗余，生成基本 E-R 图。

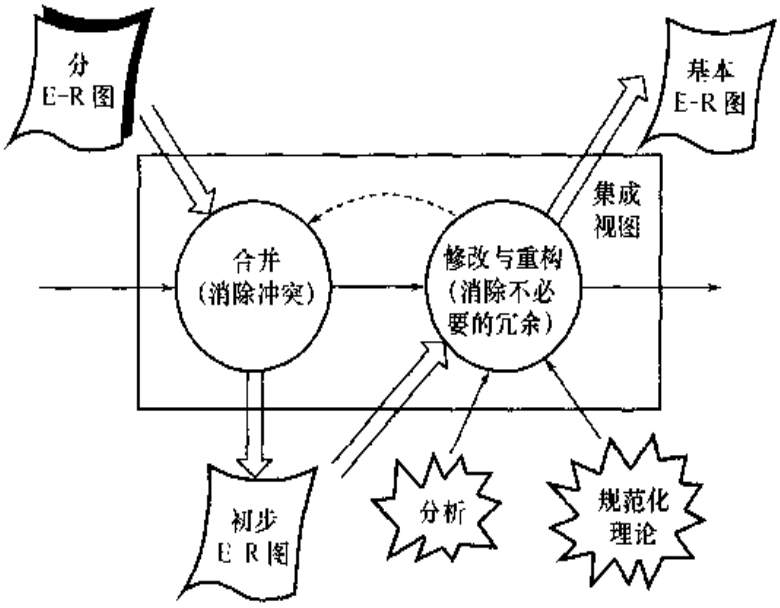


图 7.26　视图集成

一、合并分 E-R 图，生成初步 E-R 图

各个局部应用所面向的问题不同，且通常是由不同的设计人员进行局部视图设计，这就导致各个分 E-R 图之间必定会存在许多不一致的地方，称之为冲突。因此，合并分 E-R 图时并不能简单地将各个分 E-R 图画到一起，而是必须着力消除各个分 E-R 图中的不一致，以形成一个能为全系统中所有用户共同理解和接受的统一的概念模型。合理消除各分 E-R 图的冲突是合并分 E-R 图的主要工作与关键所在。

各分 E-R 图之间的冲突主要有三类：属性冲突、命名冲突和结构冲突。

1. 属性冲突

(1) 属性域冲突，即属性值的类型、取值范围或取值集合不同。例如零件号，有的部门把它定义为整数，有的部门把它定义为字符型。不同的部门对零件号的编码也不同。又如年龄，某些部门以出生日期形式表示职工的年龄，而另一些部门用整数表示职工的年龄。

(2) 属性取值单位冲突。例如，零件的重量有的以公斤为单位，有的以斤为单位，有的以克为单位。

属性冲突理论上好解决，但实际上需要各部门讨论协商，解决起来并非易事。

2. 命名冲突

(1) 同名异义，即不同意义的对象在不同的局部应用中具有相同的名字。

(2) 异名同义(一义多名)，即同一意义的对象在不同的局部应用中具有不同的名字。

如对科研项目，财务科称为项目，科研处称为课题，生产管理处称为工程。

命名冲突可能发生在实体、联系一级上，也可能发生在属性一级上。其中属性的命名冲突更为常见。处理命名冲突通常也像处理属性冲突一样，通过讨论、协商等行政手段加以解决。

3. 结构冲突

(1) 同一对象在不同应用中具有不同的抽象。例如，职工在某一局部应用中被当作实体，而在另一局部应用中则被当作属性。

解决方法通常是把属性变换为实体或把实体变换为属性，使同一对象具有相同的抽象。但变换时仍要遵循 7.3.2 节中讲述的两个准则。

(2) 同一实体在不同分 E-R 图中所包含的属性个数和属性排列次序不完全相同。

这是很常见的一类冲突，原因是不同的局部应用关心的是该实体的不同侧面。解决方法是使该实体的属性取各分 E-R 图中属性的并集，再适当调整属性的次序。

实体间的联系在不同的分 E-R 图中为不同的类型，如实体 E1 与 E2 在一个分 E-R 图中是多对多联系，在另一个分 E-R 图中是一对多联系；又如在一个分 E-R 图中 E1 与 E2 发生联系，而在另一个分 E-R 图中 E1、E2、E3 三者之间有联系。

解决方法是根据应用的语义对实体联系的类型进行综合或调整。

例如，图 7.27 中零件与产品之间存在多对多的联系——“构成”，产品、零件与供应商三者之间还存在多对多的联系——“供应”，这两个联系互相不能包含，在合并两个分 E-R 图时就应把它们综合起来。

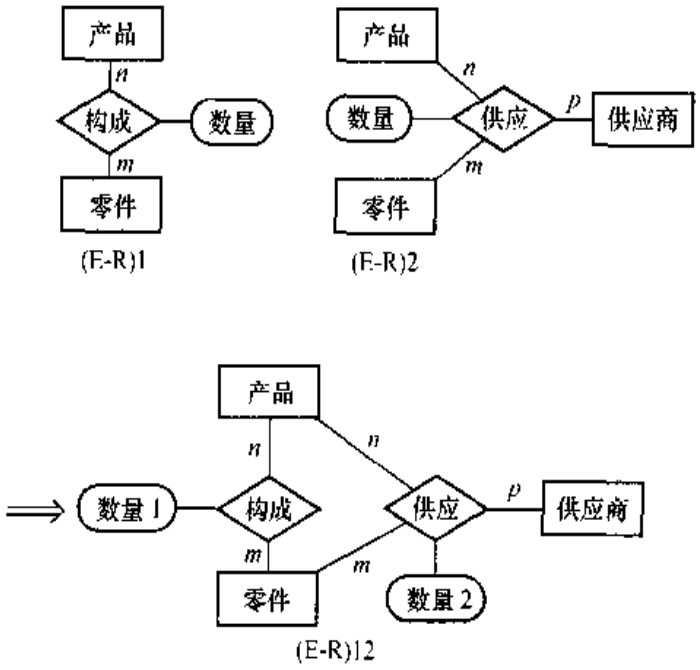


图 7.27 合并两个分 E-R 图时的综合

二、消除不必要的冗余，设计基本 E-R 图

在初步 E-R 图中，可能存在一些冗余的数据和实体间冗余的联系。所谓冗余的数据是指可由基本数据导出的数据，冗余的联系是指可由其他联系导出的联系。冗余数据和冗余联系容易破坏数据库的完整性，给数据库维护增加困难，应当予以消除。消除了冗余后的初步 E-R 图称为基本 E-R 图。

消除冗余主要采用分析方法，即以数据字典和数据流图为依据，根据数据字典中关于数据项之间逻辑关系的说明来消除冗余。如图 7.28 中，$Q_3 = Q_1 \times Q_2$，$Q_4 = \sum Q_5$。所以 Q_3 和 Q_4 是冗余数据，可以消去。并且由于 Q_3 消去，产品与材料间 $m:n$ 的冗余联系也应消去。

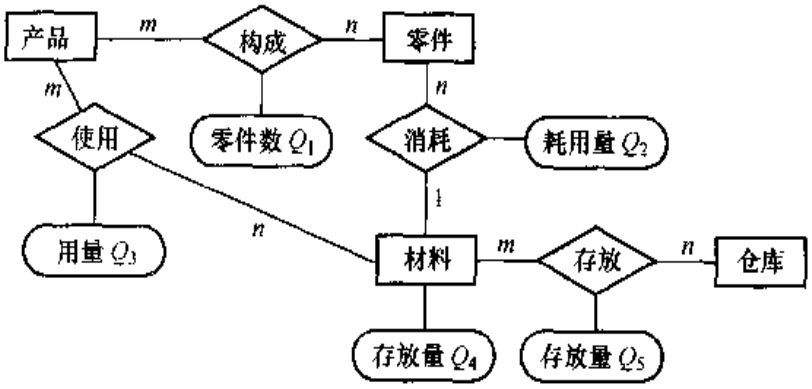


图 7.28　消除冗余

但并不是所有的冗余数据与冗余联系都必须加以消除，有时为了提高效率，不得不以冗余信息作为代价。因此在设计数据库概念结构时，哪些冗余信息必须消除，哪些冗余信息允许存在，需要根据用户的整体需求来确定。如果人为地保留了一些冗余数据，则应把数据字典中数据关联的说明作为完整性约束条件。

例如，若物种部门经常要查询各种材料的库存量，如果每次都要查询每个仓库中此种材料的库存，再对它们求和，查询效率就太低了。所以应保留 Q_4，同时把 $Q_4 = \sum Q_5$ 定义为 Q_4 的完整性约束条件。每当 Q_5 修改后，就触发该完整性检查例程，对 Q_4 作相应的修改。

除分析方法外，还可以用规范化理论来消除冗余。在规范化理论中，函数依赖的概念提供了消除冗余联系的形式化工具。具体方法如下：

1. 确定分 E-R 图实体之间的数据依赖。实体之间一对一、一对多、多对多的联系可以用实体码之间的函数依赖来表示。如图 7.29 中：

部门和职工之间一对多的联系可表示为职工号→部门号

职工和产品之间多对多的联系可表示为(职工号，产品号)→工作天数等。

于是有函数依赖集 F_L。

2. 求 F_L 的最小覆盖 G_L，差集为

$$D = F_L - G_L。$$

逐一考察 D 中的函数依赖，确定是否是冗余的联系，若是，就把它去掉。由于规范化理论受

到泛关系假设的限制，应注意下面两个问题：

（1）冗余的联系一定在 D 中，而 D 中的联系不一定是冗余的；

（2）当实体之间存在多种联系时，要将实体之间的联系在形式上加以区分。如图 7.30 中部门和职工之间另一个一对一的联系就要表示为

负责人.职工号→部门号，部门号→负责人.职工号

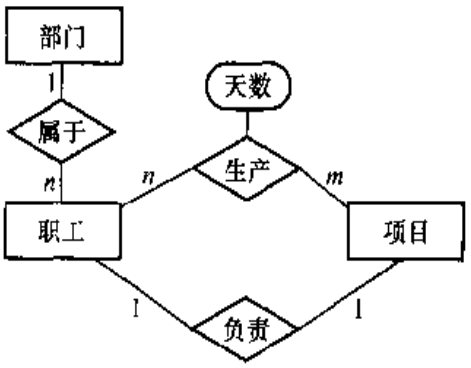


图 7.29 劳动人事管理的分 E-R 图

[**实例**] 某工厂管理信息系统的视图集成。

图 1.14(b)、图 7.24、图 7.29 分别为该厂物资、销售和劳动人事管理的分 E-R 图。图 7.30 为该系统的基本 E-R 图。基本 E-R 图中各实体的属性因篇幅有限这里从略。集成过程中，解决了以下问题：

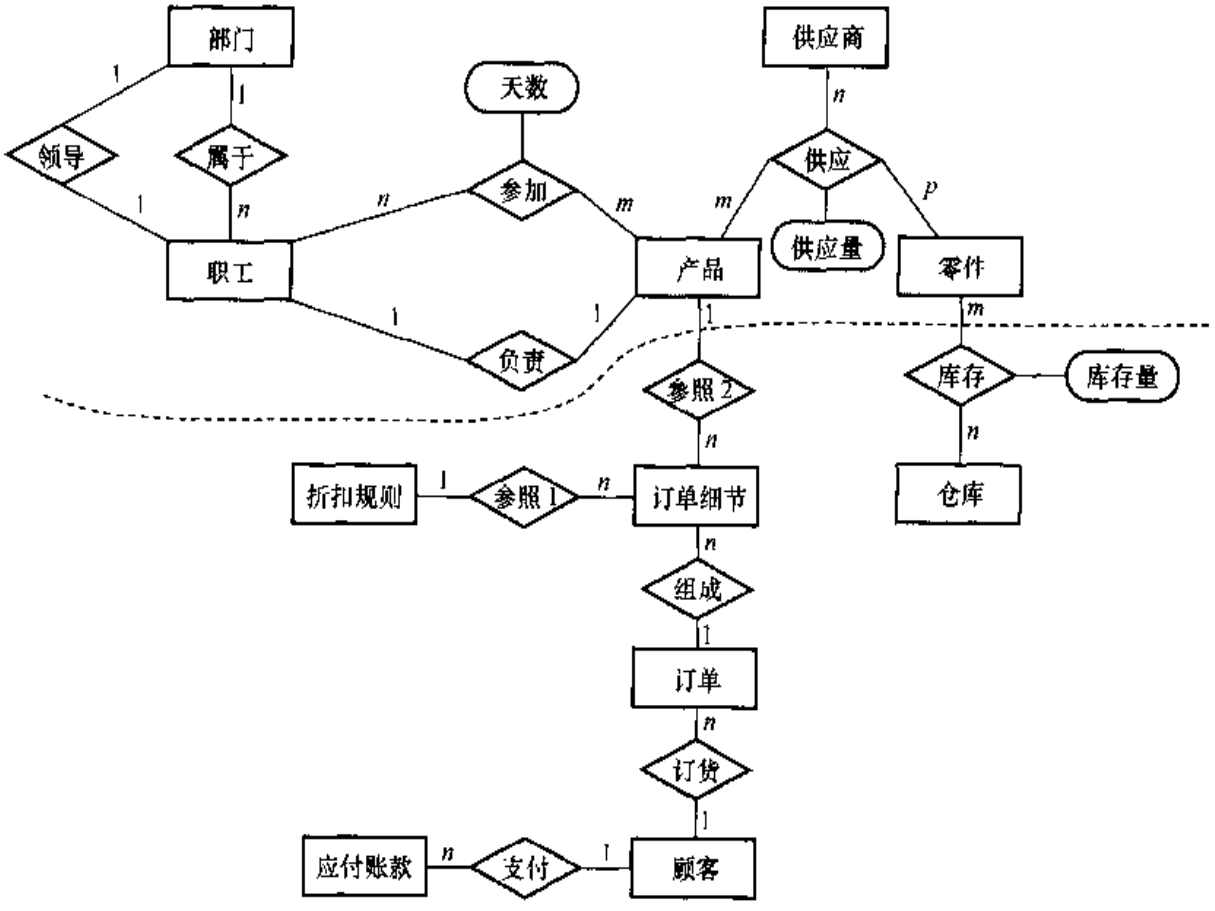


图 7.30 某工厂管理信息系统的基本 E-R 图

（1）异名同义，项目和产品含义相同。某个项目实质上是指某个产品的生产。统一用产品作实体名。

(2) 库存管理中职工与仓库的工作关系已包含在劳动人事管理的部门与职工之间的联系之中,所以可以取消。职工之间领导与被领导关系可由部门与职工(经理)之间的领导关系、部门与职工之间的从属关系两者导出,所以也可以取消。

7.4　逻辑结构设计

概念结构是独立于任何一种数据模型的信息结构。逻辑结构设计的任务就是把概念结构设计阶段设计好的基本 E-R 图转换为与选用 DBMS 产品所支持的数据模型相符合的逻辑结构。

从理论上讲,设计逻辑结构应该选择最适于相应概念结构的数据模型,然后对支持这种数据模型的各种 DBMS 进行比较,从中选出最合适的 DBMS。但实际情况往往是已给定了某种 DBMS,设计人员没有选择的余地。目前 DBMS 产品一般支持关系、网状、层次三种模型中的某一种。对某一种数据模型,各个机器系统又有许多不同的限制,提供不同的环境与工具。所以设计逻辑结构时一般要分 3 步进行(如图 7.31 所示):

(1) 将概念结构转换为一般的关系、网状、层次模型;

(2) 将转换来的关系、网状、层次模型向特定 DBMS 支持下的数据模型转换;

(3) 对数据模型进行优化。

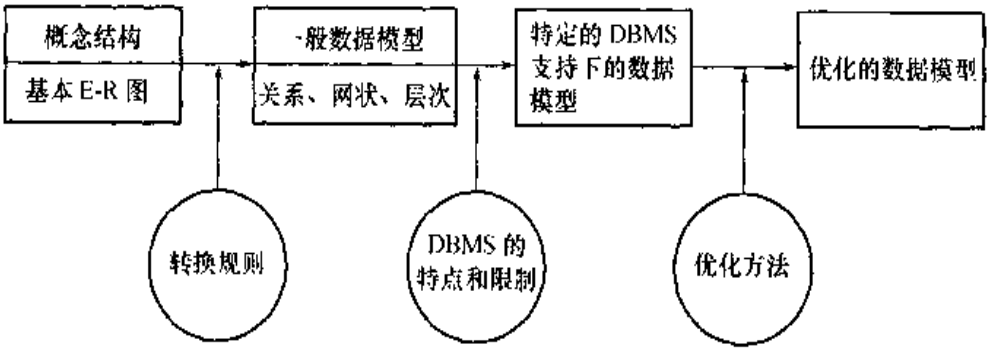


图 7.31　逻辑结构设计时的 3 个步骤

目前新设计的数据库应用系统大都采用支持关系数据模型的 RDBMS,所以这里只介绍 E-R 图向关系数据模型的转换原则与方法。

7.4.1　E-R 图向关系模型的转换

E-R 图向关系模型的转换要解决的问题是如何将实体型和实体间的联系转换为关系模式,如何确定这些关系模式的属性和码。

关系模型的逻辑结构是一组关系模式的集合。E-R 图则是由实体型、实体的属性和实体型之间的联系 3 个要素组成的。所以将 E-R 图转换为关系模型实际上就是要将实体型、实体的属性和实体型之间的联系转换为关系模式,这种转换一般遵循如下原则:

一个实体型转换为一个关系模式。实体的属性就是关系的属性，实体的码就是关系的码。

对于实体型间的联系则有以下不同的情况：

(1) 一个 1:1 联系可以转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并。如果转换为一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，每个实体的码均是该关系的候选码。如果与某一端实体对应的关系模式合并，则需要在该关系模式的属性中加入另一个关系模式的码和联系本身的属性。

(2) 一个 1:n 联系可以转换为一个独立的关系模式，也可以与 n 端对应的关系模式合并。如果转换为一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为 n 端实体的码。

(3) 一个 m:n 联系转换为一个关系模式。与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，各实体的码组成关系的码或关系码的一部分。

(4) 3 个或 3 个以上实体间的一个多元联系可以转换为一个关系模式。与该多元联系相连的各实体的码以及联系本身的属性均转换为关系的属性，各实体的码组成关系的码或关系码的一部分。

(5) 具有相同码的关系模式可合并。

下面把图 7.30 中虚线上部的 E-R 图转换为关系模型。关系的码用下横线标出。

部门(部门号，部门名，经理的职工号，…)

此为部门实体对应的关系模式。该关系模式已包含了联系“领导”所对应的关系模式。经理的职工号是关系的候选码。

职工(职工号、部门号，职工名，职务，…)

此为职工实体对应的关系模式。该关系模式已包含了联系“属于”所对应的关系模式。

产品(产品号，产品名，产品组长的职工号，…)

此为产品实体对应的关系模式。

供应商(供应商号，姓名，…)

此为供应商实体对应的关系模式。

零件(零件号，零件名，…)

此为零件实体对应的关系模式。

职工工作(职工号，产品号，工作天数，…)

此为联系“参加”所对应的关系模式。

供应(产品号，供应商号，零件号，供应量)

此为联系“供应”所对应的关系模式。

形成了一般的数据模型后，下一步就是向特定的 RDBMS 的模型转换。对于目前大都 RDBMS 来说，一般不必转换。

7.4.2 数据模型的优化

数据库逻辑设计的结果不是唯一的。为了进一步提高数据库应用系统的性能，还应该根据应用需要适当地修改、调整数据模型的结构，这就是数据模型的优化。关系数据模型的优化通常以规范化理论为指导，方法为：

1. 确定数据依赖。在7.2.3“数据字典”一节中已讲到用数据依赖的概念分析和表示数据项之间的联系，写出每个数据项之间的数据依赖。按需求分析阶段所得到的语义，分别写出每个关系模式内部各属性之间的数据依赖以及不同关系模式属性之间的数据依赖。

2. 对于各个关系模式之间的数据依赖进行极小化处理，消除冗余的联系，具体方法已在7.3.4小节中讲解。

3. 按照数据依赖的理论对关系模式逐一进行分析，考察是否存在部分函数依赖、传递函数依赖、多值依赖等，确定各关系模式分别属于第几范式。

4. 按照需求分析阶段得到的处理要求，分析对于这样的应用环境这些模式是否合适，确定是否要对某些模式进行合并或分解。

必须注意的是，并不是规范化程度越高的关系就越优。例如，当查询经常涉及到两个或多个关系模式的属性时，系统经常进行连接运算。连接运算的代价是相当高的，可以说关系模型低效的主要原因就是连接运算引起的。这时可以考虑将这几个关系合并为一个关系。因此在这种情况下，第二范式甚至第一范式也许是合适的。

又如，非BCNF的关系模式虽然从理论上分析会存在不同程度的更新异常或冗余，但如果在实际应用中对此关系模式只是查询，并不执行更新操作，则就不会产生实际影响。所以对于一个具体应用来说，到底规范化到什么程度，需要权衡响应时间和潜在问题两者的利弊决定。

5. 对关系模式进行必要的分解，提高数据操作的效率和存储空间的利用率。常用的两种分解方法是水平分解和垂直分解。

水平分解是把(基本)关系的元组分为若干子集合，定义每个子集合为一个子关系，以提高系统的效率。根据“80/20原则”，一个大关系中，经常被使用的数据只是关系的一部分，约20%，可以把经常使用的数据分解出来，形成一个子关系。如果关系R上具有n个事务，而且多数事务存取的数据不相交，则R可分解为少于或等于n个子关系，使每个事务存取的数据对应一个关系。

垂直分解是把关系模式R的属性分解为若干子集合，形成若干子关系模式。垂直分解的原则是，经常在一起使用的属性从R中分解出来形成一个子关系模式。垂直分解可以提高某些事务的效率，但也可能使另一些事务不得不执行连接操作，从而降低了效率。因此是否进行垂直分解取决于分解后R上的所有事务的总效率是否得到了提高。垂直分解需要确保无损连接性和保持函数依赖，即保证分解后的关系具有无损连接性和保持函数依赖性。

这可以用第五章中的模式分解算法对需要分解的关系模式进行分解和检查。

规范化理论为数据库设计人员判断关系模式优劣提供了理论标准，可用来预测模式可能出现的问题，使数据库设计工作有了严格的理论基础。

7.4.3 设计用户子模式

将概念模型转换为全局逻辑模型后，还应该根据局部应用需求，结合具体 DBMS 的特点，设计用户的外模式。

目前关系数据库管理系统一般都提供了视图(View)概念，可以利用这一功能设计更符合局部用户需要的用户外模式。

定义数据库全局模式主要是从系统的时间效率、空间效率、易维护等角度出发。由于用户外模式与模式是相对独立的，因此在定义用户外模式时可以注重考虑用户的习惯与方便。包括：

(1) 使用更符合用户习惯的别名。

在合并各分 E-R 图时，曾做了消除命名冲突的工作，以使数据库系统中同一关系和属性具有唯一的名字。这在设计数据库整体结构时是非常必要的。用 View 机制可以在设计用户 View 时重新定义某些属性名，使其与用户习惯一致，以方便使用。

(2) 可以对不同级别的用户定义不同的 View，以保证系统的安全性。

假设有关系模式产品(产品号，产品名，规格，单价，生产车间，生产负责人，产品成本，产品合格率，质量等级)，可以在产品关系上建立两个视图：

为一般顾客建立视图：

产品 1(产品号，产品名，规格，单价)

为产品销售部门建立图：

产品 2(产品号，产品名，规格，单价，车间，生产负责人)

顾客视图中只包含允许顾客查询的属性；销售部门视图中只包含允许销售部门查询的属性；生产领导部门则可以查询全部产品数据。这样就可以防止用户非法访问本来不允许他们查询的数据，保证了系统的安全性。

(3) 简化用户对系统的使用。

如果某些局部应用中经常要使用某些很复杂的查询，为了方便用户，可以将这些复杂查询定义为视图，用户每次只对定义好的视图进行查询，大大简化了用户的使用。

7.5 数据库的物理设计

数据库在物理设备上的存储结构与存取方法称为数据库的物理结构，它依赖于选定的数据库管理系统。为一个给定的逻辑数据模型选取一个最适合应用要求的物理结构的过

程,就是数据库的物理设计。

数据库的物理设计通常分为两步:

(1) 确定数据库的物理结构,在关系数据库中主要指存取方法和存储结构;

(2) 对物理结构进行评价,评价的重点是时间和空间效率。

如果评价结果满足原设计要求,则可进入到物理实施阶段,否则,就需要重新设计或修改物理结构,有时甚至要返回逻辑设计阶段修改数据模型。

7.5.1 数据库物理设计的内容和方法

不同的数据库产品所提供的物理环境、存取方法和存储结构有很大差别,能供设计人员使用的设计变量、参数范围也很不相同,因此没有通用的物理设计方法可遵循,只能给出一般的设计内容和原则。希望设计优化的物理数据库结构,使得在数据库上运行的各种事务响应时间小、存储空间利用率高、事务吞吐率大。为此,首先对要运行的事务进行详细分析,获得选择物理数据库设计所需要的参数。其次,要充分了解所用 RDBMS 的内部特征,特别是系统提供的存取方法和存储结构。

对于数据库查询事务,需要得到如下信息:

- 查询的关系;
- 查询条件所涉及的属性;
- 连接条件所涉及的属性;
- 查询的投影属性。

对于数据更新事务,需要得到如下信息:

- 被更新的关系;
- 每个关系上的更新操作条件所涉及的属性;
- 修改操作要改变的属性值。

除此之外,还需要知道每个事务在各关系上运行的频率和性能要求。例如,事务 T 必须在 10 秒钟内结束,这对于存取方法的选择具有重大影响。

上述这些信息是确定关系的存取方法的依据。

应注意的是,数据库上运行的事务会不断变化、增加或减少,以后需要根据上述设计信息的变化调整数据库的物理结构。

通常关系数据库物理设计的内容主要包括 :

- 为关系模式选择存取方法;
- 设计关系、索引等数据库文件的物理存储结构。

下面就介绍这些设计内容和方法。

7.5.2 关系模式存取方法选择

数据库系统是多用户共享的系统,对同一个关系要建立多条存取路径才能满足多用户

的多种应用要求。物理设计的任务之一就是要确定选择哪些存取方法，即建立哪些存取路径。

存取方法是快速存取数据库中数据的技术。数据库管理系统一般都提供多种存取方法。常用的存取方法有三类。第一类是索引方法，目前主要是 B + 树索引方法；第二类是聚簇(Cluster)方法；第三类是 HASH 方法。

B + 树索引方法是数据库中经典的存取方法，使用最普遍。

一、索引存取方法的选择

所谓选择索引存取方法实际上就是根据应用要求确定对关系的哪些属性列建立索引、哪些属性列建立组合索引、哪些索引要设计为唯一索引等。一般来说：

1. 如果一个(或一组)属性经常在查询条件中出现，则考虑在这个(或这组)属性上建立索引(或组合索引)；

2. 如果一个属性经常作为最大值和最小值等聚集函数的参数，则考虑在这个属性上建立索引；

3. 如果一个(或一组)属性经常在连接操作的连接条件中出现，则考虑在这个(或这组)属性上建立索引；

关系上定义的索引数并不是越多越好，系统为维护索引要付出代价，查找索引也要付出代价。例如，若一个关系的更新频率很高，这个关系上定义的索引数不能太多。因为更新一个关系时，必须对这个关系上有关的索引做相应的修改。

二、聚簇存取方法的选择

为了提高某个属性(或属性组)的查询速度，把这个或这些属性(称为**聚簇码**)上具有相同值的元组集中存放在连续的物理块称为聚簇。

聚簇功能可以大大提高按聚簇码进行查询的效率。例如，要查询信息系的所有学生名单，设信息系有 500 名学生，在极端情况下，这 500 名学生所对应的数据元组分布在 500 个不同的物理块上。尽管对学生关系已按所在系建有索引，由索引很快找到了信息系学生的元组标识，避免了全表扫描，然而再由元组标识去访问数据块时就要存取 500 个物理块，执行 500 次 I/O 操作。如果将同一系的学生元组集中存放，则每读一个物理块可得到多个满足查询条件的元组，从而显著地减少了访问磁盘的次数。

聚簇功能不但适用于单个关系，也适用于经常进行连接操作的多个关系。即把多个连接关系的元组按连接属性值聚集存放，聚簇中的连接属性称为**聚簇码**。这就相当于把多个关系按“预连接”的形式存放，从而大大提高连接操作的效率。

一个数据库可以建立多个聚簇，一个关系只能加入一个聚簇。

选择聚簇存取方法，即确定需要建立多少个聚簇，每个聚簇中包括哪些关系。

下面先设计候选聚簇，一般来说：

(1) 对经常在一起进行连接操作的关系可以建立聚簇；

(2) 如果一个关系的一组属性经常出现在相等比较条件中,则该单个关系可建立聚簇;

(3) 如果一个关系的一个(或一组)属性上的值重复率很高,则此单个关系可建立聚簇。即对应每个聚簇码值的平均元组数不太少。太少了,聚簇的效果不明显。

然后检查候选聚簇中的关系,取消其中不必要的关系。

(1) 从聚簇中删除经常进行全表扫描的关系;

(2) 从聚簇中删除更新操作远多于连接操作的关系;

(3) 不同的聚簇中可能包含相同的关系,一个关系可以在某一个聚簇中,但不能同时加入多个聚簇。要从这多个聚簇方案(包括不建立聚簇)中选择一个较优的,即在这个聚簇上运行各种事务的总代价最小。

必须强调的是,聚簇只能提高某些应用的性能,而且建立与维护聚簇的开销是相当大的。对已有关系建立聚簇,将导致关系中元组移动其物理存储位置,并使此关系上原有的索引无效,必须重建。当一个元组的聚簇码值改变时,该元组的存储位置也要做相应移动,聚簇码值要相对稳定,以减少修改聚簇码值所引起的维护开销。

因此,当通过聚簇码进行访问或连接是该关系的主要应用,与聚簇码无关的其他访问很少或者是次要的,这时可以使用聚簇。尤其当 SQL 语句中包含有与聚簇码有关的 ORDER BY、GROUP BY、UNION、DISTINCT 等子句或短语时,使用聚簇特别有利,可以省去对结果集的排序操作;否则很可能会适得其反。

三、HASH 存取方法的选择

有些数据库管理系统提供了 HASH 存取方法。选择 HASH 存取方法的规则如下:

如果一个关系的属性主要出现在等值连接条件中或主要出现在相等比较选择条件中,而且满足下列两个条件之一,则此关系可以选择 HASH 存取方法:

(1) 如果一个关系的大小可预知,而且不变;

(2) 如果关系的大小动态改变,而且数据库管理系统提供了动态 HASH 存取方法。

7.5.3 确定数据库的存储结构

确定数据库物理结构主要指确定数据的存放位置和存储结构,包括:确定关系、索引、聚簇、日志、备份等的存储安排和存储结构,确定系统配置等。

确定数据的存放位置和存储结构要综合考虑存取时间、存储空间利用率和维护代价 3 个方面的因素。这 3 个方面常常是相互矛盾的,因此需要进行权衡,选择一个折中方案。

1. 确定数据的存放位置

为了提高系统性能,应该根据应用情况将数据的易变部分与稳定部分、经常存取部分和存取频率较低部分分开存放。

例如,目前许多计算机有多个磁盘或磁盘阵列,因此可以将表和索引放在不同的磁盘上,在查询时,由于磁盘驱动器并行工作,可以提高物理 I/O 读写的效率;也可以将比较大的

表分放在两个磁盘上,以加快存取速度,这在多用户环境下特别有效;还可以将日志文件与数据库对象(表、索引等)放在不同的磁盘上,以改进系统的性能。

由于各个系统所能提供的对数据进行物理安排的手段、方法差异很大,因此设计人员应仔细了解给定的 RDBMS 提供的方法和参数,针对应用环境的要求,对数据进行适当的物理安排。

2. 确定系统配置

DBMS 产品一般都提供了一些系统配置变量、存储分配参数,供设计人员和 DBA 对数据库进行物理优化。初始情况下,系统都为这些变量赋予了合理的默认值。但是这些值不一定适合每一种应用环境,在进行物理设计时,需要重新对这些变量赋值,以改善系统的性能。

系统配置变量很多,例如:同时使用数据库的用户数,同时打开的数据库对象数,内存分配参数,缓冲区分配参数(使用的缓冲区长度、个数),存储分配参数,物理块的大小,物理块装填因子,时间片大小,数据库的大小,锁的数目等。这些参数值影响存取时间和存储空间的分配,在物理设计时就要根据应用环境确定这些参数值,以使系统性能最佳。

在物理设计时对系统配置变量的调整只是初步的,在系统运行时还要根据系统实际运行情况做进一步的调整,以期切实改进系统性能。

7.5.4 评价物理结构

数据库物理设计过程中需要对时间效率、空间效率、维护代价和各种用户要求进行权衡,其结果可以产生多种方案。数据库设计人员必须对这些方案进行细致的评价,从中选择一个较优的方案作为数据库的物理结构。

评价物理数据库的方法完全依赖于所选用的 DBMS,主要是从定量估算各种方案的存储空间、存取时间和维护代价入手,对估算结果进行权衡、比较,选择出一个较优的合理的物理结构。如果该结构不符合用户需求,则需要修改设计。

7.6 数据库的实施和维护

完成数据库的物理设计之后,设计人员就要用 RDBMS 提供的数据定义语言和其他实用程序将数据库逻辑设计和物理设计结果严格描述出来,成为 DBMS 可以接受的源代码,再经过调试产生目标模式。然后就可以组织数据入库了,这就是数据库实施阶段。

7.6.1 数据的载入和应用程序的调试

数据库实施阶段包括两项重要的工作,一项是数据的载入,另一项是应用程序的编码和调试。

一般数据库系统中,数据量都很大,而且数据来源于部门中的各个不同的单位,数据的

组织方式、结构和格式都与新设计的数据库系统有相当的差距。组织数据录入就要将各类源数据从各个局部应用中抽取出来，输入计算机，再分类转换，最后综合成符合新设计的数据库结构的形式，输入数据库。因此这样的数据转换、组织入库的工作是相当费力、费时的。

特别是原系统是手工数据处理系统时，各类数据分散在各种不同的原始表格、凭证、单据之中。在向新的数据库系统中输入数据时，还要处理大量的纸质文件，工作量就更大。

为提高数据输入工作的效率和质量，应该针对具体的应用环境设计一个数据录入子系统，由计算机来完成数据入库的任务。在源数据入库之前要采用多种方法对它们进行检验，以防止不正确的数据入库，这部分的工作在整个数据输入子系统中是非常重要的。

现有的 RDBMS 一般都提供不同 RDBMS 之间数据转换的工具，若原来是数据库系统，就要充分利用新系统的数据转换工具。

数据库应用程序的设计应该与数据库设计同时进行，因此在组织数据入库的同时还要调试应用程序。应用程序的设计、编码和调试的方法、步骤在软件工程等课程中有详细讲解，这里就不赘述了。

7.6.2 数据库的试运行

在原有系统的数据有一小部分已输入数据库后，就可以开始对数据库系统进行联合调试，这又称为数据库的试运行。

这一阶段要实际运行数据库应用程序，执行对数据库的各种操作，测试应用程序的功能是否满足设计要求。如果不满足，对应用程序部分则要修改、调整，直到达到设计要求为止。

在数据库试运行时，还要测试系统的性能指标，分析其是否达到设计目标。在对数据库进行物理设计时已初步确定了系统的物理参数值，但一般的情况下，设计时的考虑在许多方面只是近似的估计，和实际系统运行总有一定的差距，因此必须在试运行阶段实际测量和评价系统性能指标。事实上，有些参数的最佳值往往是经过运行调试后找到的。如果测试的结果与设计目标不符，则要返回物理设计阶段，重新调整物理结构，修改系统参数，某些情况下甚至要返回逻辑设计阶段，修改逻辑结构。

这里特别要强调两点。第一，上面已经讲到组织数据入库是十分费时、费力的事，如果试运行后还要修改数据库的设计，还要重新组织数据入库。因此应分期分批地组织数据入库，先输入小批量数据做调试用，待试运行基本合格后，再大批量输入数据，逐步增加数据量，逐步完成运行评价。第二，在数据库试运行阶段，由于系统还不稳定，硬、软件故障随时都可能发生。而系统的操作人员对新系统还不熟悉，误操作也不可避免，因此应首先调试运行 DBMS 的恢复功能，做好数据库的转储和恢复工作。一旦故障发生，能使数据库尽快恢复，尽量减少对数据库的破坏。

7.6.3 数据库的运行和维护

数据库试运行合格后，数据库开发工作就基本完成，即可投入正式运行了。但是，由于应用环境在不断变化，数据库运行过程中物理存储也会不断变化，对数据库设计进行评价、调整、修改等维护工作是一个长期的任务，也是设计工作的继续和提高。

在数据库运行阶段，对数据库经常性的维护工作主要是由 DBA 完成的，它包括：

1. 数据库的转储和恢复

数据库的转储和恢复是系统正式运行后最重要的维护工作之一。DBA 要针对不同的应用要求制定不同的转储计划，以保证一旦发生故障能尽快将数据库恢复到某种一致的状态，并尽可能减少对数据库的破坏。

2. 数据库的安全性、完整性控制

在数据库运行过程中，由于应用环境的变化，对安全性的要求也会发生变化，比如有的数据原来是机密的，现在是可以公开查询的了，而新加入的数据又可能是机密的了。系统中用户的密级也会改变。这些都需要 DBA 根据实际情况修改原有的安全性控制。同样，数据库的完整性约束条件也会变化，也需要 DBA 不断修正，以满足用户要求。

3. 数据库性能的监督、分析和改造

在数据库运行过程中，监督系统运行，对监测数据进行分析，找出改进系统性能的方法是 DBA 的又一重要任务。目前有些 DBMS 产品提供了监测系统性能参数的工具，DBA 可以利用这些工具方便地得到系统运行过程中一系列性能参数的值。DBA 应仔细分析这些数据，判断当前系统运行状况是否是最佳，应当做哪些改进，例如调整系统物理参数或对数据库进行重组织或重构造等。

4. 数据库的重组织与重构造

数据库运行一段时间后，由于记录不断增、删、改，会使数据库的物理存储情况变坏，降低了数据的存取效率，数据库性能下降，这时 DBA 就要对数据库进行重组织或部分重组织（只对频繁增、删的表进行重组织）。DBMS 一般都提供数据重组织用的实用程序。在重组织的过程中，按原设计要求重新安排存储位置、回收垃圾、减少指针链等，提高系统性能。

数据库的重组织并不修改原设计的逻辑和物理结构，而数据库的重构造则不同，它是指部分修改数据库的模式和内模式。

由于数据库应用环境发生变化，增加了新的应用或新的实体，取消了某些应用，有的实体与实体间的联系也发生了变化等，使原有的数据库设计不能满足新的需求，需要调整数据库的模式和内模式。例如，在表中增加或删除某些数据项，改变数据项的类型，增加或删除某个表，改变数据库的容量，增加或删除某些索引等。当然数据库的重构也是有限的，只能做部分修改。如果应用变化太大，重构也无济于事，说明此数据库应用系统的生命周期已经结束，应该设计新的数据库应用系统了。

7.7 小 结

数据库设计这一章主要讨论数据库设计的方法和步骤，列举了较多的实例，详细介绍了数据库设计各个阶段的目标、方法、应注意的事项。其中的重点是概念结构的设计和逻辑结构的设计，这也是数据库设计过程中最重要的两个环节。

学习这一章，要努力掌握书中讨论的基本方法，还要能在实际工作中运用这些思想，设计符合应用需求的数据库应用系统。

习 题

1. 试述数据库设计过程。
2. 试述数据库设计过程的各个阶段上的设计描述。
3. 试述数据库设计过程中形成的数据库模式。
4. 试述数据库设计的特点。
5. 需求分析阶段的设计目标是什么？调查的内容是什么？
6. 数据字典的内容和作用是什么？
7. 什么是数据库的概念结构？试述其特点和设计策略。
8. 什么叫数据抽象？试举例说明。
9. 试述数据库概念结构设计的重要性和设计步骤。
10. 为什么要视图集成？视图集成的方法是什么？
11. 什么是数据库的逻辑结构设计？试述其设计步骤。
12. 试把第一章习题 12 和习题 13 中的 E-R 图转换为关系模型。
13. 试用规范化理论中有关范式的概念分析习题 12 中你设计的关系模型中各个关系模式的候选码，它们属于第几范式？会产生什么更新异常？
14. 规范化理论对数据库设计有什么指导意义？
15. 试述数据库物理设计的内容和步骤。
16. 数据输入在实施阶段的重要性是什么？如何保证输入数据的正确性？
17. 什么是数据库的再组织和重构造？为什么要进行数据库的再组织和重构造？
18. 现有一局部应用，包括两个实体："出版社"和"作者"，这两个实体是多对多的联系，请读者自己设计适当的属性，画出 E-R 图，再将其转换为关系模型（包括关系名、属性名、码和完整性约束条件）。
19. 请设计一个图书馆数据库，此数据库中对每个借阅者保存读者记录，包括：读者号，姓名，地址，性别，年龄，单位；对每本书存有：书号，书名，作者，出版社；对每本被借出的书存有读者号、借出日期和应还日期。要求：给出 E-R 图，再将其转换为关系模型。

课程大作业《数据库设计与应用开发》

1. 基本要求：在某个 RDBMS 软件基础上，利用合适的应用系统开发工具为某个部门或单位开发一个

数据库应用系统

2. 实验目的：

- 通过实践，掌握本章介绍的数据库设计方法。
- 学会在一个实际的 RDBMS 软件平台上创建数据库。
- 培养团队合作精神，要求 5～6 位同学组成一个开发小组，每位同学承担不同角色（例如：项目管理员、DBA、系统分析员、系统设计员、系统开发员、系统测试员）。

3. 内容与具体要求：

- 给出数据库设计各个阶段的详细设计报告；
- 写出系统的主要功能和使用说明；
- 提交运行的系统；
- 写出收获和体会，包括已解决和尚未解决的问题，进一步完善的设想与建议；
- 每个小组进行 60 分钟的报告和答辩，讲解设计方案，演示系统运行，汇报分工与合作情况。

本章参考文献

1 Wiederhold G. Database Design(Second Edition). McGraw-Hill,1983

（本书是一本综合性教科书，覆盖了数据库设计的各个阶段，重点强调了物理设计阶段。）

2 Albano A(ed.). Computer-Aided Database Design:The DATAID Project. North-Holland,1985

（本书讲解了数据库设计及计算机辅助数据库设计的方法和工具。）

3 Yao S B(ed.). Principles of Database Design:Logical Organization. Vol. 1,Prentice-Hall,1985

（本书收集了从需求说明技术到模式调整等方面的研究成果。）

4 Yao S B(ed.). 数据库设计Ⅰ. 罗大卫，刘化明译. 计算机工程与应用，第七期，1985

5 姚卿达. 数据库设计. 北京：高等教育出版社，1987

（本书系统讲解了数据库设计的方法和工具，覆盖了数据库设计的所有阶段，提出了适合中小型数据库设计的多级方法 DBD/MLM，并设计了一个处理要求模型化工具 PSSB 作为处理模式的描述语言，把结构和处理设计相结合。）

6 王珊，冯念真. 计算机应用系统的设计和开发. 北京：高等教育出版社，1989

（本书从应用系统的开发和研制的角度讲解了数据库设计。）

7 Chen P P S. The Entity-Relationship Model:Towards a Unified View of Data. ACM TODS Vol. 1,No. 1,1976

（Peter Chen 在本文中首次提出了著名的实体－联系模型，在以后的论文中他对 E-R 模型作了深入的研究。）

8 Chen P P S. The Entity-Relationship Model:A Basis for the Enterprise View of Data. AFIPS Conf. Proc. Vol. 46,AFIPS Press,NJ,1977

9 Chen P P S. The Entity-Relationship Approach to Logical DataBase Design. Q. E. D. MonographSeries,1977

10 Chen P P S. Recent Literature of the Entity-Relationship Approach. Proceedings of International Conference on Entity-Relationship Approach to Systems Analysis and Design,1979

11 Smith J,Smith D. Database Abstractions: Aggregation and Generalization. TODS,2:2,1977

（本文在 E-R 模型中引入了概括(Generalization)和聚集(Aggregation)的概念。）

12 Hammer M, McLeod D. Database Description with SDM: A Semantic Data Model. TODS, 6:3, 1980
（本文提出了扩展的 E－R 模型，引进了超类/子类等概念。）

13 Elmasri R, Weeldreyer J, Hevner A. The Category Concept: An Extension to the Entity-Relationship Model. International Journal on Data and Knowledge Engineering, 1:1, 1985
（本文对 E-R 模型进行了扩充。）

14 王珊，吴鸥琦. E-R 图/数据模型转换的一点注记. 小型微型计算机系统，1983
（本文研究了从 E-R 模型向数据模型转换的问题。）

15 萨师煊，王珊. N-E-R Model: A New Method to Design Enterprise Schema. 第一届国际计算机应用大会论文集，1984

16 Schkolnick M. A Survey of Physical Database Design Methodology and Techniques. in Proceedings of VLDB, 1978
（本文综述了物理数据库设计方法。）

17 Yeh R T et al. Software and Data Base Engineering: Towards a Common Design Methodology. Issues in Data Base Management, 1979
（本文指出数据库设计是一项庞大的软件工程。）

18 萨师煊. 数据库的理论与实践. 计算机科学，1983

19 萨师煊. 数据库系统和系统分析. 计算机工程与设计，1984

20 萨师煊，王珊. 数据库设计理论和实践. 计算机应用与软件，VOL. 2, NO. 4, 1984

21 Ceri S, Navathe S, Wiederhold G. Distribution Design of Logical Database Schemas. TSE, 9:4, 1983
（本文全面讨论了分布式数据库的逻辑设计。）

22 Batini C, Lenzerini M. and Navathe S. A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys, 18:4, 1986
（本文研究和比较了数据库设计中各种模式集成方法。）

23 Navathe S, Gadgil S. A Methodology for View Integration in Logical Database Design. in Proceedings of VLDB, 1982
（本文提出了数据库设计中视图集成方法。）

第八章 数据库编程

建立数据库后就要开发应用系统了。本章讲解应用系统中如何使用编程方法对数据库进行操纵的技术。

标准 SQL 是非过程化的查询语言，具有操作统一、面向集合、功能丰富、使用简单等多项优点。但和程序设计语言相比，高度非过程化的优点同时也造成了它的一个弱点：缺少流程控制能力，难以实现应用业务中的逻辑控制，SQL 编程技术可以有效克服 SQL 语言实现复杂应用方面的不足，提高应用系统和 RDBMS 间的互操作性。

应用系统中使用 SQL 编程来访问和管理数据库中数据的方式主要有：嵌入式 SQL（Embedded SQL，ESQL）、PL/SQL（Procedural Language/SQL）、ODBC（Open Data Base Connectivity）编程、JDBC（Java Data Base Connectivity）编程和 OLEDB（Object Linking and Embedding DB）编程等方式。本章中将讲解这些编程技术的概念和方法。本章介绍嵌入式 SQL、PL/SQL 和 ODBC 编程，JDBC 和 OLEDB 编程与 ODBC 编程思路基本相同，限于篇幅本章就不讲解了。

8.1 嵌入式 SQL

本书第三章已经讲到，SQL 的特点之一是，在交互式和嵌入式两种不同的使用方式下，SQL 的语法结构基本上是一致的。当然在程序设计的环境下，SQL 语句要做某些必要的扩充。

8.1.1 嵌入式 SQL 的处理过程

嵌入式 SQL 是将 SQL 语句嵌入程序设计语言中，被嵌入的程序设计语言，如 C、C++、Java，称为宿主语言，简称主语言。

对 ESQL，RDBMS 一般采用预编译方法处理，即由 RDBMS 的预处理程序对源程序进行扫描，识别出 ESQL 语句，把它们转换成主语言调用语句，以使主语言编译程序能识别它们，然后由主语言的编译程序将纯的主语言程序编译成目标码，如图 8.1 所示。

在ESQL中，为了能够区分SQL语句与主语言语句，所有SQL语句都必须加前缀EXEC SQL，以(;)结束成为一个程序片断：

EXEC SQL <SQL语句>;



图8.1 ESQL基本处理过程

8.1.2 嵌入式SQL语句与主语言之间的通信

将SQL嵌入到高级语言中混合编程，SQL语句负责操纵数据库，高级语言语句负责控制程序流程。这时程序中会含有两种不同计算模型的语句，它们之间应该如何通信呢？

数据库工作单元与源程序工作单元之间的通信主要包括：

(1) 向主语言传递SQL语句的执行状态信息，使主语言能够据此信息控制程序流程，主要用SQL通信区(SQL Communication Area，简称SQLCA)实现；

(2) 主语言向SQL语句提供参数，主要用主变量(Host Variable)实现；

(3) 将SQL语句查询数据库的结果交主语言处理，主要用主变量和游标(Cursor)实现。

一、SQL通信区

SQL语句执行后，系统要反馈给应用程序若干信息，主要包括描述系统当前工作状态和运行环境的各种数据。这些信息将送到SQLCA中。应用程序从SQLCA中取出这些状态信息，据此决定接下来执行的语句。

SQLCA在应用程序中用EXEC SQL INCLUDE SQLCA加以定义。SQLCA中有一个变量SQLCODE，用来存放每次执行SQL语句后返回的代码。

应用程序每执行完一条SQL语句之后都应该测试一下SQLCODE的值，以了解该SQL语句执行情况并做相应处理。如果SQLCODE等于预定义的常量SUCCESS，则表示SQL语句成功，否则在SQLCODE存放错误代码。程序员可以根据错误代码查找问题。

二、主变量

嵌入式SQL语句中可以使用主语言的程序变量来输入或输出数据。SQL语句中使用的主语言程序变量简称为**主变量**。主变量根据其作用的不同，分为输入主变量和输出主变量。**输入主变量**由应用程序对其赋值，SQL语句引用；**输出主变量**由SQL语句对其赋值或设置状态信息，返回给应用程序。

一个主变量可以附带一个任选的指示变量(Indicator Variable)。**指示变量**是一个整型变量，用来"指示"所指主变量的值或条件。指示变量可以指示输入主变量是否为空值，可以检测输出主变量是否空值，值是否被截断。

所有主变量和指示变量必须在 SQL 语句 BEGIN DECLARE SECTION 与 END DECLARE SECTION 之间进行说明。说明之后,主变量可以在 SQL 语句中任何一个能够使用表达式的地方出现,为了与数据库对象名(表名、视图名、列名等)区别,SQL 语句中的主变量名和指示变量前要加冒号(:)作为标志。

三、游标

SQL 是面向集合的,一条 SQL 语句可以产生或处理多条记录。而主语言是面向记录的,一组主变量一次只能存放一条记录。所以仅使用主变量并不能完全满足 SQL 语句向应用程序输出数据的要求,为此嵌入式 SQL 引入了游标的概念,用游标来协调这两种不同的处理方式。

游标是系统为用户开设的一个数据缓冲区,存放 SQL 语句的执行结果,每个游标区都有一个名字。用户可以通过游标逐一获取记录,并赋给主变量,交由主语言进一步处理。

四、建立和关闭数据库连接

嵌入式 SQL 程序要访问数据库必须先连接数据库。RDBMS 根据用户信息对连接请求进行合法性验证,只有通过了身份验证,才能建立一个可用的合法连接。

1. 建立数据库连接

建立连接的 ESQL 语句是

EXEC SQL CONNECT TO *target*[AS *connection-name*][USER *user-name*];

其中:

target 是要连接的数据库服务器,它可以是一个常见的服务器标识串,如 < dbname > @ < hostname > : < port > 。或者是包含服务器标识的 SQL 串常量,也可以是 DEFAULT。

connect-name 是可选的连接名,连接必须是一个有效的标识符,主要用来识别一个程序内同时建立的多个连接,如果在整个程序内只有一个连接也可以不指定连接名。

如果程序运行过程中建立了多个连接,执行的所有数据库单元的工作都在该操作提交时所选择的当前连接上。程序运行过程中可以修改当前连接,对应的嵌入式 SQL 语句为

EXEC SQL SET CONNECTION *connection-name*|DEFAULT;

2. 关闭数据库连接

当某个连接上的所有数据库操作完成后,应用程序应该主动释放所占用的连接资源。关闭数据库连接的 ESQL 语句是

EXEC SQL DISCONNECT [*connection*];

其中 *connection* 是 EXEC SQL CONNECT 所建立的数据库连接。

五、程序实例

为了能够更好地理解有关概念,下面给出一个简单的 ESQL 编程实例。

[例 1] 依次检查某个系的学生记录,交互式更新某些学生年龄。

```
EXEC SQL BEGIN DECLARE SECTION;     /* 主变量说明开始 */
```

```
char deptname[20];
char HSno[9];
char HSname[20];
char HSsex[2];
int HSage;
int NEWAGE;
EXEC SQL END DECLARE SECTION;          /* 主变量说明结束 */
long SQLCODE;
EXEC SQL INCLUDE sqlca;                /* 定义 SQL 通信区 */
int main(void)                         /* C 语言主程序开始 */
{
int count = 0;
char yn;                               /* 变量 yn 代表 yes 或 no */
printf("Please choose the department name(CS/MA/IS): ");
scanf("%s", &deptname);                /* 为主变量 deptname 赋值 */
EXEC SQL CONNECT TO TEST@localhost:54321 USER
      "SYSTEM" /"MANAGER";             /* 连接数据库 TEST */
EXEC SQL DECLARE SX CURSOR FOR         /* 定义游标 SX */
    SELECT Sno, Sname, Ssex, Sage      /* SX 对应的语句 */
    FROM Student
    WHERE SDept = :deptname;
EXEC SQL OPEN SX;                      /* 打开游标 SX,指向查询结果的第一行 */
for ( ; ; )                            /* 用循环结构逐条处理结果集中的记录 */
{EXEC SQL FETCH SX INTO :HSno, :HSname, :HSsex, :HSage;
                                       /* 推进游标,将当前数据放入主变量 */
if (sqlca.sqlcode !=0)                 /* sqlcode !=0,表示操作不成功 */
  break;                               /* 利用 SQLCA 中的状态信息决定何时退出循环 */
if(count ++==0)                        /* 如果是第一行的话,先打出行头 */
  printf("\n%-10s %-20s %-10s %-10s\n", "Sno", "Sname", "Ssex", "Sage");
  printf("%-10s %-20s %-10s %-10d\n", HSno, HSname, HSsex, HSage);
                                       /* 打印查询结果 */
  printf("UPDATE AGE(y/n)?");          /* 询问用户是否要更新该学生的年龄 */
  do {
     scanf("%c",&yn);
     }
     while(yn !='N' && yn !='n' && yn !='Y' && yn !='y');
  if (yn =='y' || yn =='Y')            /* 如果选择更新操作 */
  {
   printf("INPUT NEW AGE:");
   scanf("%d",&NEWAGE);                /* 用户输入新年龄到主变量中 */
   EXEC SQL UPDATE Student             /* 嵌入式 SQL 更新语句 */
            SET Sage = :NEWAGE
            WHERE CURRENT OF SX ;
```

```
    |                                    /* 对当前游标指向的学生年龄进行更新 */
  }
EXEC SQL CLOSE SX;                     /* 关闭游标 SX 不再和查询结果对应 */
EXEC SQL COMMIT WORK;                  /* 提交更新 */
EXEC SQL DISCONNECT TEST;              /* 断开数据库连接 */
|
```

8.1.3 不用游标的 SQL 语句

有的 ESQL SQL 语句不需要使用游标。它们是：说明性语句、数据定义语句、数据控制语句、查询结果为单记录的 SELECT 语句、非 CURRENT 形式的增删改语句。

一、查询结果为单记录的 SELECT 语句

这类语句不需要使用游标，因为查询结果只有一个，只需要用 INTO 子句指定存放查询结果的主变量。

[例 2] 根据学生号码查询学生信息。假设已经把要查询的学生的学号赋给了主变量 givensno。

```
EXEC SQL SELECT Sno,Sname,Ssex,Sage,Sdept
         INTO :Hsno,:Hname,:Hsex,:Hage,:Hdept
         FROM Student
         WHERE Sno = :givensno;
```

使用单记录的 SELECT 语句需要注意以下几点：

(1) INTO 子句、WHERE 子句和 HAVING 短语的条件表达式中均可以使用主变量。

(2) 查询结果为空值的处理。

查询返回的记录中，可能某些列为空值 NULL。为了表示空值，在 INTO 子句的主变量后面跟有指示变量，当查询得出的某个数据项为空值时，系统会自动将相应主变量后面的指示变量置为负值，而不再向该主变量赋值。所以当指示变量值为负值时，不管主变量为何值，均认为主变量值为 NULL。

指示变量只能用于 INTO 子句中。

(3) 如果查询结果实际上并不是单条记录，而是多条记录，则程序出错，RDBMS 会在 SQLCA 中返回错误信息。

[例 3] 查询某个学生选修某门课程的成绩。假设已经把将要查询的学生的学号赋给了主变量 givensno，将课程号赋给了主变量 givencno。

```
EXEC SQL SELECT Sno,Cno,Grade
         INTO :Hsno,:Hcno,:Hgrade:Gradeid          /* 指示变量 Gradeid */
         FROM SC
         WHERE Sno = :givensno AND Cno = :givencno;
```

如果 Gradeid <0,则不论 Hgrade 为何值,均认为该学生成绩为空值。

二、非 CURRENT 形式的增删改语句

有些增删改语句不需要使用游标,不是 CURRENT 形式的。在 UPDATE 的 SET 子句和 WHERE 子句中可以使用主变量,SET 子句还可以使用指示变量。

[例 4] 修改某个学生选修 1 号课程的成绩。

```
EXEC SQL UPDATE SC
          SET Grade = :newgrade                    /* 修改的成绩已赋给主变量 */
          WHERE Sno = :givensno;                   /* 学号已赋给主变量 givensno */
```

[例 5] 将计算机系全体学生年龄置 NULL 值。

```
Sageid = -1;
EXEC SQL UPDATE Student
          SET Sage = :Raise :Sageid
          WHERE Sdept ='CS';
```

将指示变量 Sageid 赋一个负值后,无论主变量 Raise 为何值,RDBMS 都会将 CS 系所有学生的年龄置空值。它等价于:

```
EXEC SQL UPDATE Student
          SET Sage = NULL
          WHERE Sdept ='CS';
```

[例 6] 某个学生退学了,现要将有关他的所有选课记录删除掉。假设该学生的姓名已赋给主变量 stdname。

```
EXEC SQL DELETE
          FROM SC
          WHERE Sno =
               (SELECT Sno
                FROM Student
                WHERE Sname = :stdname);
```

[例 7] 某个学生新选修了某门课程,将有关记录插入 SC 表中。假设插入的学号已赋给主变量 stdno,课程号已赋给主变量 couno。

```
gradeid = -1;                                      /* 用作指示变量,赋为负值 */
EXEC SQL INSERT
          INTO SC(Sno,Cno,Grade)
          VALUES(:stdno,:couno,:gr :gradeid);
```

由于该学生刚选修课程,成绩应为空,所以要把指示变量赋为负值。

8.1.4 使用游标的 SQL 语句

必须使用游标的 SQL 语句有:查询结果为多条记录的 SELECT 语句、CURRENT 形式的 UPDATE 和 DELETE 语句。

一、查询结果为多条记录的 SELECT 语句

一般情况下,SELECT 语句查询结果是多条记录,因此需要用游标机制,将多条记录一次一条送主程序处理,从而把对集合的操作转换为对单个记录的处理。使用游标的步骤为:

(1) 说明游标。用 DECLARE 语句为一条 SELECT 语句定义游标:

```
EXEC SQL DECLARE <游标名> CURSOR FOR <SELECT 语句>;
```

定义游标仅仅是一条说明性语句,这时 RDBMS 并不执行 SELECT 语句。

(2) 打开游标。用 OPEN 语句将定义的游标打开。

```
EXEC SQL OPEN <游标名>;
```

打开游标实际上是执行相应的 SELECT 语句,把查询结果取到缓冲区中。这时游标处于活动状态,指针指向查询结果集中的第一条记录。

(3) 推进游标指针并取当前记录。

```
EXEC SQL FETCH <游标名>
    INTO <主变量>[ <指示变量>][,<主变量>[ <指示变量>]]…;
```

其中主变量必须与 SELECT 语句中的目标列表达式具有一一对应关系。

用 FETCH 语句把游标指针向前推进一条记录,同时将缓冲区中的当前记录取出来送至主变量供主语言进一步处理。通过循环执行 FETCH 语句逐条取出结果集中的行进行处理。

(4) 关闭游标。用 CLOSE 语句关闭游标,释放结果集占用的缓冲区及其他资源。

```
EXEC SQL CLOSE <游标名>;
```

游标被关闭后,就不再和原来的查询结果集相联系。但被关闭的游标可以再次被打开,与新的查询结果相联系。

二、CURRENT 形式的 UPDATE 和 DELETE 语句

UPDATE 语句和 DELETE 语句都是集合操作,如果只想修改或删除其中某个记录,则需要用带游标的 SELECT 语句查出所有满足条件的记录,从中进一步找出要修改或删除的记录,然后用 CURRENT 形式的 UPDATE 和 DELETE 语句修改或删除之。即 UPDATE 语句和 DELETE 语句中要用子句

```
WHERE CURRENT OF <游标名>
```

来表示修改或删除的是最近一次取出的记录,即游标指针指向的记录。

[**例 1**]中的 UPDATE 就是用 CURRENT 形式的。

注意:当游标定义中的 SELECT 语句带有 UNION 或 ORDER BY 子句时,或者该 SELECT 语句相当于定义了一个不可更新的视图时,不能使用 CURRENT 形式的 UPDATE 语句和 DELETE 语句。

8.1.5　动态 SQL

前面所讲的嵌入式 SQL 语句中使用的主变量、查询目标列、条件等都是固定的，属于静态 SQL 语句。静态嵌入式 SQL 语句能够满足一般要求，但某些应用可能要到执行时才能够确定要提交的 SQL 语句，查询的条件。就要使用动态 SQL 来解决这类问题。

动态 SQL 方法允许在程序运行过程中临时"组装"SQL 语句。动态 SQL 支持动态组装 SQL 语句和动态参数两种形式，给开发者提供设计任意 SQL 语句的能力。

一、使用 SQL 语句主变量

程序主变量包含的内容是 SQL 语句的内容，而不是原来保存数据的输入或输出变量，这样的变量称为 SQL 语句主变量。SQL 语句主变量在程序执行期间可以设定不同的 SQL 语句，然后立即执行。

[例 9]　创建基本表 TEST。

```
EXEC SQL BEGIN DECLARE SECTION;
const char * stmt = "CREATE TABLE test(a int);";          /* SQL 语句主变量 */
EXEC SQL END DECLARE SECTION;
   ... ...
EXEC SQL EXECUTE IMMEDIATE :stmt;                         /* 执行语句 */
```

二、动态参数

动态参数是 SQL 语句中的可变元素，使用参数符号(?)表示该位置的数据在运行时设定。和前面使用的主变量不同：动态参数的输入不是编译时完成绑定，而是通过 (prepare) 语句准备主变量和执行(execute)时绑定数据或主变量来完成。使用动态参数的步骤：

1. 声明 SQL 语句主变量。

变量的 SQL 内容包含动态参数(?)。

2. 准备 SQL 语句(PREPARE)。

PREPARE 将分析含主变量的 SQL 语句内容，建立语句中包含的动态参数的内部描述符，并用 <语句名 > 标识它们的整体。

```
EXEC SQL PREPARE <语句名> FROM <SQL 语句主变量>;
```

3. 执行准备好的语句(EXECUTE)。

EXECUTE 将 SQL 语句中分析出的动态参数和主变量或数据常量绑定作为语句的输入或输出变量。

```
EXEC SQL EXECUTE <语句名> [INTO <主变量表>][USING <主变量或常量>];
```

[例 10]　向 TEST 中插入元组。

```
EXEC SQL BEGIN DECLARE SECTION;
const char * stmt = "INSERT INTO test VALUES(?);";        /* 声明 SQL 主变量 */
EXEC SQL END DECLARE SECTION;
```

```
... ...
EXEC SQL PREPARE mystmt FROM :stmt;                    /* 准备语句 */
... ...
EXEC SQL EXECUTE mystmt USING 100;                     /* 执行语句 */
EXEC SQL EXECUTE mystmt USING 200;                     /* 执行语句 */
```

8.1.6 小结

这一节主要讲解嵌入式 SQL，讲解如何把 SQL 语句嵌入到某种高级语言中，例如 C 语言中，利用高级语言的强大计算能力来实现复杂应用的需求。

在嵌入式 SQL 中，SQL 语句用来与数据库打交道，存取数据库中的数据。主语言语句用来控制程序流程以及对取出的数据做进一步加工处理。

SQL 与主语言具有不同的数据处理方式。SQL 是面向集合的，而主语言是面向记录的。所以，嵌入式 SQL 引入了游标的概念，用游标来协调这两种不同的处理方式。学习这一章，要掌握游标的概念，更加重要的是能够在实际安装的 RDBMS 上通过编程的方式用嵌入 SQL 对数据库进行操作，用游标来编写实际的应用程序。

8.2 存储过程

SQL99 标准中给出了 SQL-invoked routines 的概念。SQL-invoked routines 可以分为存储过程(SQL-invoked procedure)和函数(SQL-invoked function)两类。下面介绍存储过程。

建立存储过程可以指定所使用的程序设计语言。**PL/SQL**(Procedural Language/ SQL，PL/SQL)是编写数据库存储过程的一种过程语言。它结合了 SQL 的数据操作能力和过程化语言的流程控制能力，是 SQL 的过程化扩展。下面首先介绍 PL/SQL，然后介绍存储过程。

8.2.1 PL/SQL 的块结构

基本的 SQL 是高度非过程化的语言。ESQL 将 SQL 语句嵌入程序设计语言，借助高级语言的控制功能实现过程化。PL/SQL 是对 SQL 的扩展，使其增加了过程化语句功能。

PL/SQL 程序的基本结构是块。所有的 PL/SQL 程序都是由块组成的。这些块之间可以互相嵌套，每个块完成一个逻辑操作。图 8.2 是 PL/SQL 块的基本结构。

8.2.2 变量常量的定义

1. PL/SQL 中定义变量的语法形式是：

变量名 数据类型 [[NOT NULL]: = 初值表达式]或
变量名 数据类型 [[NOT NULL]初值表达式]

2. 常量的定义类似于变量的定义：

常量名 数据类型 CONSTANT : = 常量表达式

常量必须要给一个值，并且该值在存在期间或常量的作用域内不能改变。如果试图修改它，PL/SQL 将返回一个异常。

3. 赋值语句

变量名称：=表达式

```
定义部分{ DECLARE                          /* 定义的变量、常量等只能在该基本块中使用 */
          ------ 变量、常量、游标、异常等    /* 当基本块执行结束时，定义就不再存在 */
执行部分{ BEGIN
          ------ SQL 语句、PL/SQL 的流程控制语句
          EXCEPTION                        /* 遇到不能继续执行的情况称为异常 */
          ------ 异常处理部分               /* 在出现异常时，采取措施来纠正错误或报告错误 */
          END;
```

图 8.2　PL/SQL 块的基本结构

8.2.3　控制结构

PL/SQL 提供了流程控制语句，主要有条件控制语句和循环控制语句。这些语句的语法、语义和一般的高级语言（如 C 语言）类似，这里只做概要的介绍。读者使用时要参考具体的产品手册的语法规则。

一、条件控制语句

一般有三种形式的 IF 语句：IF-THEN，IF-THEN-ELSE 和嵌套的 IF 语句。

1. IF *condition* THEN

```
  Sequence_of_statements;        /* 条件为真时语句序列才被执行 */
END IF                           /* 条件为假或 NULL 时什么也不做，控制转移至下一个语句 */
```

2. IF *condition* THEN

```
  Sequence_of_statements1;       /* 条件为真时执行语句序列 1 */
ELSE
  Sequence_of_statements2;       /* 条件为假或 NULL 时执行语句序列 2 */
END IF;
```

3. 在 THEN 和 ELSE 子句中还可以再包括 IF 语句，即 IF 语句可以嵌套。

二、循环控制语句

PL/SQL 有三种循环结构：LOOP，WHILE-LOOP 和 FOR-LOOP。

1. 最简单的循环语句 LOOP

```
LOOP
  Sequence_of_statements;        /* 循环体，一组 PL/SQL 语句 */
END LOOP;
```

多数数据库服务器的 PL/SQL 都提供 EXIT、BREAK 或 LEAVE 等循环结束语句，以保证 LOOP 语句块能够在适当的条件下结束。

2. WHILE-LOOP

```
WHILE condition LOOP
  Sequence_of_statements;
END LOOP;
```

每次执行循环体语句之前，首先对条件进行求值。如果条件为真，则执行循环体内的语句序列。如果条件为假，则跳过循环并把控制传递给下一个语句。

3. FOR-LOOP

```
FOR count IN [REVERSE]bound1 … bound2 LOOP
  Sequence_of_statements;
END LOOP;
```

FOR 循环的基本执行过程是：将 *count* 设置为循环的下界 *bound1*，检查它是否小于上界 bound 2。当指定 REVERSE 时则将 count 设置为循环的上界 bound 2，检查 count 是否大于下界 bound 1。如果越界则执行跳出循环，否则执行循环体，然后按照步长（+1 或 -1）更新 *count* 的值，重新判断条件。

三、错误处理

如果 PL/SQL 在执行时出现异常，则应该让程序在产生异常的语句处停下来，根据异常的类型去执行异常处理语句。

SQL 标准对数据库服务器提供什么样的异常处理做出了建议，要求 PL/SQL 管理器提供完善的异常处理机制。相对于 ESQL 简单的提供执行状态信息 SQLCODE，这里的异常处理就复杂多了。读者要根据具体系统的支持情况来进行错误处理。

8.2.4 存储过程

PL/SQL 块主要有两种类型，即命名块和匿名块。前面介绍的是匿名块。匿名块每次执行时都要进行编译，它不能被存储到数据库中，也不能在其他的 PL/SQL 块中调用。存储过程和函数是命名块，它们被编译后保存在数据库中，可以被反复调用，运行速度较快。

一、存储过程的优点

存储过程是由 PL/SQL 语句书写的过程，这个过程经编译和优化后存储在数据库服务器中，因此称它为存储过程，使用时只要调用即可。

使用存储过程具有以下优点：

1. 由于存储过程不像解释执行的 SQL 语句那样在提出操作请求时才进行语法分析和优化工作，因而运行效率高，它提供了在服务器端快速执行 SQL 语句的有效途径。

2. 存储过程降低了客户机和服务器之间的通信量。客户机上的应用程序只要通过网络向服务器发出存储过程的名字和参数，就可以让 RDBMS 执行许多条的 SQL 语句，并执行数据处理。只有最终处理结果才返回客户端。

3. 方便实施企业规则。可以把企业规则的运算程序写成存储过程放入数据库服务器中，由 RDBMS 管理，既有利于集中控制，又能够方便地进行维护。当用户规则发生变化时只要修改存储过程，无须修改其他应用程序。

二、存储过程的用户接口

用户通过下面的 SQL 语句创建、重新命名、执行和删除存储过程：

1. 创建存储过程

```
CREATE Procedure 过程名([参数1,参数2,...])   /* 存储过程首部 */
AS
<PL/SQL 块>;                                  /* 存储过程体,描述该存储过程的操作 */
```

存储过程包括过程首部和过程体。

过程名：是数据库服务器合法的对象标识。

参数列表：用名字来标识调用时给出的参数值，必须指定值的数据类型。存储过程的参数也可以定义输入参数、输出参数或输入/输出参数。默认为输入参数。

过程体：是一个 <PL/SQL 块>。包括声明部分和可执行语句部分。<PL/SQL 块> 的基本结构已经在 8.2.1 和 8.2.2 中讲解了。

[例 11] 利用存储过程来实现下面的应用：从一个账户转指定数额的款项到另一个账户中。

```
CREATE PROCEDURE TRANSFER(inAccount INT, outAccount INT, amount FLOAT)
  AS DECLARE
       totalDeposit FLOAT;
  BEGIN                                  /* 检查转出账户的余额 */
    SELECT total INTO totalDeposit FROM ACCOUNT WHERE ACCOUNTNUM = outAccount;
    IF totalDeposit IS NULL THEN         /* 账户不存在或账户中没有存款 */
         ROLLBACK;
         RETURN;
    END IF;
    IF totalDeposit < amount THEN        /* 账户账户存款不足 */
         ROLLBACK;
         RETURN;
    END IF;
    UPDATE account SET total = total - amount WHERE ACCOUNTNUM = outAccount;
                                         /* 修改转出账户,减去转出额 */
  UPDATE account SET total = total + amount WHERE ACCOUNTNUM = inAccount;
                                         /* 修改转入账户,增加转出额 */
```

```
    COMMIT;                                    /* 提交转账事务 */
END;
```

可以使用 ALTER Proccdure 重命名一个存储过程：

```
ALTER Procedure 过程名 1 RENAME TO 过程名 2;
```

2. 执行存储过程

```
CALL/PERFORM Procedure 过程名([参数 1,参数 2,...]);
```

使用 CALL 或者 PERFORM 等方式激活存储过程的执行。在 PL/SQL 中，数据库服务器支持在过程体中调用其他存储过程。

[例 12] 从账户 01003815868 转一万元到 01003813828 账户中。

```
CALL Procedure TRANSFER(01003815868, 01003813828, 10000);
```

3. 删除存储过程

```
DROP PROCEDURE 过程名();
```

三、游标

和嵌入式 SQL 一样，在 PL/SQL 中如果 SELECT 语句只返回一条记录，可以将该结果存放到变量中。当查询返回多条记录时，就要使用游标对结果集进行处理。一个游标与一个 SQL 语句相关联。

游标的概念已经在 8.1 节中介绍过了。不同的是 PL/SQL 存储过程中的语句格式稍有差别，另外 PL/SQL 中的游标由 PL/SQL 引擎管理，可以具有更多内置扩展特性。

8.2.5 小结

本节讲解了 SQL 的过程化扩展 PL/SQL 和存储过程。存储过程经编译和优化后存储在数据库服务器中，因而运行效率高，可以降低客户机和服务器之间的通信量，有利于集中控制，又能够方便地进行维护。正是由于存储过程具有这些优点而被广泛使用。

由于各个 RDBMS 产品提供的 PL/SQL 和存储过程的语法不尽相同，读者在使用时要认真阅读产品手册。

8.3 ODBC 编程

这一节介绍如何使用 ODBC 来进行数据库应用程序的设计，使用 ODBC 编写的应用程序可移植性好，能同时访问不同的数据库，共享多个数据资源。

8.3.1 数据库互连概述

提出和产生 ODBC 的原因是不同的数据库管理系统的存在。

目前广泛使用的 RDBMS 有多种，尽管这些系统都属于关系数据库，也都遵循 SQL 标

准,但是不同的系统有许多差异。因此,在某个 RDBMS 下编写的应用程序就不能在另一个 RDBMS 下运行,适应性和可移植性较差。例如,运行在 SYBASE 上的应用系统想在 ORACLE 上运行就必须进行修改移植。这种修改移植是费事的,开发人员必须清楚了解不同 RDBMS 的确切区别,细心地一一进行修改、测试。

更加重要的是,许多应用程序需要共享多个部门的数据资源,访问不同的 RDBMS。为此,人们开始研究和开发连接不同 RDBMS 的方法、技术和软件,使数据库系统"开放",能够"数据库互连"。其中 ODBC,就是为了解决这样的问题而由微软公司推出的产品。ODBC 是微软公司开放服务体系(Windows Open Services Architecture,WOSA)中有关数据库的一个组成部分,它建立了一组规范,并提供一组访问数据库的标准 API。作为规范它具有两重功效或约束力:一方面规范应用开发,另一方面规范 RDBMS 应用接口。

8.3.2 ODBC 工作原理概述

使用 ODBC 开发应用系统,体系结构如图 8.3 所示,它由四部分构成:用户应用程序、驱动程序管理器(ODBC Driver Manager)、数据库驱动程序(ODBC Driver)、数据源(如 RDBMS 和数据库)。

一、应用程序

应用程序提供用户界面、应用逻辑和事务逻辑。使用 ODBC 开发数据库应用程序时,应用程序调用的是标准的 ODBC 函数和 SQL 语句。应用层使用 ODBC API 调用接口与数据库进行交互。使用 ODBC 来开发应用系统的程序简称为 ODBC 应用程序,包括的内容有:

- 请求连接数据库;
- 向数据源发送 SQL 语句;
- 为 SQL 语句执行结果分配存储空间,定义所读取的数据格式;
- 获取数据库操作结果,或处理错误;
- 进行数据处理并向用户提交处理结果;
- 请求事务的提交和回滚操作;
- 断开与数据源的连接。

二、驱动程序管理器

驱动程序管理器是用来管理各种驱动程序的。

驱动程序管理器由微软公司提供,它包含在 ODBC32. DLL 中,对用户是透明的。它管理应用程序和驱动程序之间的通信。驱动程序管理器的主要功能包括装载 ODBC 驱动程序、选择和连接正确的驱动程序、管理数据源、检查 ODBC 调用参数的合法性及记录 ODBC 函数的调用等,当应用层需要时返回驱动程序的有关信息。

ODBC 驱动程序管理器可以建立、配置或删除数据源,并查看系统当前所安装的数据库 ODBC 驱动程序。

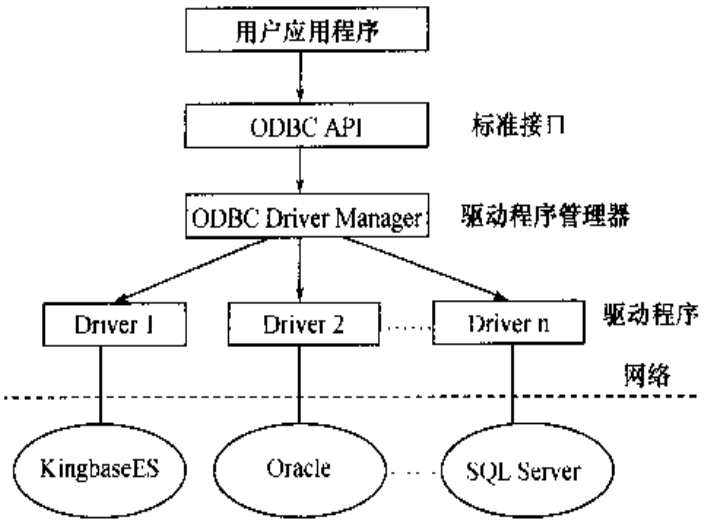


图 8.3 ODBC 应用系统的体系结构

三、数据库驱动程序

ODBC 通过驱动程序来提供应用系统与数据库平台的独立性。

ODBC 应用程序不能直接存取数据库，其各种操作请求由驱动程序管理器提交给某个 RDBMS 的 ODBC 驱动程序，通过调用驱动程序所支持的函数来存取数据库。数据库的操作结果也通过驱动程序返回给应用程序。如果应用程序要操纵不同的数据库，就要动态地链接到不同的驱动程序上。

目前的 ODBC 驱动程序主要有单束和多束两类。单束一般是数据源和应用程序在同一台机器上，驱动程序直接完成对数据文件的 I/O 操作，这时驱动程序相当于数据管理器。多束驱动程序支持客户机/服务器、客户机/应用服务器/数据库服务器等网络环境下的数据访问，这时由驱动程序完成数据库访问请求的提交和结果集接收，应用程序使用驱动程序提供的结果集管理接口操纵执行后的结果数据。

四、ODBC 数据源管理

数据源是最终用户需要访问的数据，包含了数据库位置和数据库类型等信息，实际上是一种数据连接的抽象。

ODBC 给每个被访问的数据源指定唯一的，数据源名（Data Source Name，简称 DSN），并映射到所有必要的、用来存取数据的低层软件。在连接中，用数据源名来代表用户名、服务器名、所连接的数据库名等。最终用户无需知道 DBMS 或其他数据管理软件、网络以及有关 ODBC 驱动程序的细节，数据源对最终用户是透明的。

例如，假设某个学校在 MS SQL Server 和 KingbaseES 上创建了两个数据库：学校人事数据库和教学科研数据库。学校的信息系统要从这两个数据库中存取数据。为了方便的与两个数据库连接，为学校人事数据库创建一个数据源名 PERSON，PERSON 就是一个 DSN。同样，为教学科研数据库创建一个名为 EDU 的数据源。此后，当要访问每一个数据库时，只要

与 PERSON 和 EDU 连接即可，不需要记住使用的驱动程序、服务器名称、数据库名等。所以在开发 ODBC 数据库应用程序时首先要建立数据源并给它命名。

8.3.3 ODBC API 基础

各个数据库厂商的 ODBC 应用程序接口(DDBC Application Intertace，ODBC API)都要符合两方面的一致性：

- API 一致性，API 一致性级别有核心级、扩展 1 级、扩展 2 级；
- 语法 一致性，语法一致性级别有最低限度 SQL 语法级、核心 SQL 语法级、扩展 SQL 语法级。

一、函数概述

ODBC 3.0 标准提供了 76 个函数接口，大致可以分为：

- 分配和释放环境句柄、连接句柄、语句句柄；
- 连接函数(SQLDriverconnect 等)；
- 与信息相关的函数(如获取描述信息函数 SQLGetinfo、SQLCetFuction)；
- 事务处理函数(如 SQLEndTran)；
- 执行相关函数(SQLExecdirect、SQLExecute 等)；
- 编目函数，ODBC 3.0 提供了 11 个编目函数如 SQLTables、SQLColumn 等。应用程序可以通过对编目函数的调用来获取数据字典的信息如权限、表结构等。

注意：ODBC 1.0 和 ODBC 2.x、ODBC 3.x 函数使用上的差异，很多函数在 3.x 上已经被替换或丢弃了。因此，必须注意使用的版本带来的不同之处。

也可以使用 MFC 的 ODBC 类。因为 MFC ODBC 对较复杂的 ODBC API 进行了封装，提供了简化的调用接口，大大方便了数据库应用程序的开发。不必了解 ODBC API 和 SQL 的具体细节，利用 ODBC 类即可完成对数据库的大部分操作。

二、句柄及其属性

句柄是 32 位整数值，代表一个指针。ODBC 3.0 中句柄可以分为环境句柄、连接句柄、语句句柄或描述符句柄四类，对于每种句柄不同的驱动程序有不同的数据结构，这四种句柄的关系如图 8.4 所示。

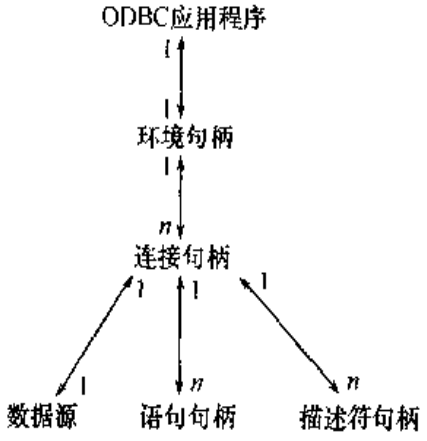


图 8.4 应用程序句柄之间的关系

1. 每个 ODBC 应用程序需要建立一个 ODBC 环境，分配一个环境句柄，存取数据的全局性背景如环境状态、当前环境状态诊断、当前在环境上分配的连接句柄等；

2. 一个环境句柄可以建立多个连接句柄，每一个连接句柄实现与一个数据源之间的连接；

3. 在一个连接中可以建立多个语句句柄，它不只是

一个 SQL 语句，还包括 SQL 语句产生的结果集以及相关的信息等；

4. 在 ODBC 3.0 中又提出了描述符句柄的概念，它是描述 SQL 语句的参数、结果集列的元数据集合。

三、数据类型

ODBC 定义了两套数据类型，即 SQL 数据类型和 C 数据类型。SQL 数据类型用于数据源，而 C 数据类型用于应用程序的 C 代码。它们之间的转换情况如表 8.1 所示。应用程序可以通过 SQLGetTypeInfo 来获取不同的驱动程序对于数据类型的支持情况。

表 8.1 SQL 数据类型和 C 数据类型之间的转换规则

	SQL 数据类型	C 数据类型
SQL 数据类型	数据源之间转换	应用程序变量传送到语句参数(SQLBindparameter)
C 数据类型	从结果集列中返回到应用程序变量(SQLBindcol)	应用程序变量之间转换

8.3.4 ODBC 的工作流程

使用 ODBC 的应用系统大致的工作流程，从开始配置数据源到回收各种句柄，如图 8.5 所示。下面将结合具体的应用实例来介绍如何使用 ODBC 开发应用系统。

[例 13] 将 KingbaseES 数据库中 Student 表的数据备份到 SQL SERVER 数据库中。

该应用涉及两个不同的 RDBMS 中的数据源，因此使用 ODBC 来开发应用程序，只要改变应用程序中连接函数(SQLConnect)的参数，就可以连接不同 RDBMS 的驱动程序，连接两个数据源。

在应用程序运行前，已经在 KingbaseES 和 SQL SERVER 中分别建立了 STUDENT 关系表：

```
CREATE TABLE Student
  (Sno CHAR(9) PRIMARY KEY,   /* 列级完整性约束条
                                 件，Sno 是主码 */
   Sname CHAR(20) UNQUE       /* Sname 取唯一值 */
   Ssex CHAR(2),
   Sage SMALLINT,
   Sdept CHAR(20)
  );
```

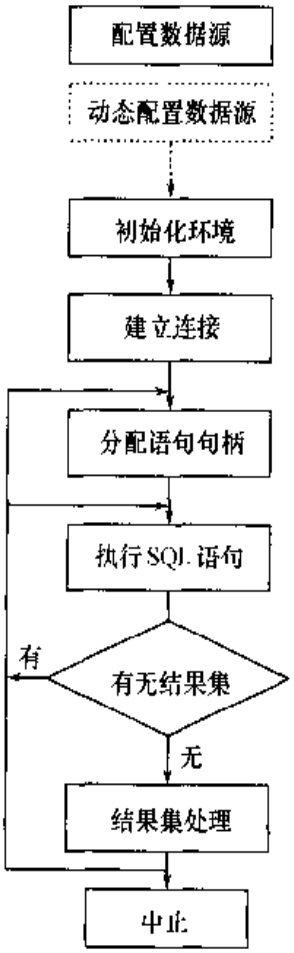


图 8.5 ODBC 的工作流程

应用程序要执行的操作是:在 KingbaseES 上执行 SELECT * FROM STUDENT;把获取的结果集,通过多次执行 INSERT INTO STUDENT (Sno,Sname,Ssex,Sage,Sddept) VALUES (?, ?, ?, ?, ?); 插入到 SQL SERVER 的 STUDENT 表中。

一、配置数据源

配置数据源有两种方法:(1)运行数据源管理工具来进行配置;(2)使用 Driver Manager 提供的 ConfigDsn 函数来增加、修改或删除数据源。这种方法特别适用于在应用程序中创建的临时使用的数据源。

在[**例 13**]中,采用了第一种方法创建数据源。因为要同时用到 KingbaseES 和 SQL Server,所以分别建立两个数据源,将其取名为 KingbaseES ODBC 和 SQLServer。不同的驱动器厂商提供了不同的配置数据源界面,建立这两个数据源的具体步骤从略。

读者可以参考具体产品使用指南。本书附带光盘中的实验报告样例给出了[**例 13**]创建数据源的详细过程。

程序的源码

```
#include <stdlib.h>
#include <stdio.h>
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <Sqltypes.h>

#define SNO_LEN 30
#define NAME_LEN 50
#define DEPART_LEN 100
#define SSEX_LEN 5

int main()
{
 /* Step 1 定义句柄和变量 */
 //以 king 开头的表示的是连接 KingbaseES 的变量
 //以 server 开头的表示的是连接 SQLSERVER 的变量
SQLHENV     kinghenv, serverhenv;                    //环境句柄
SQLHDBC     kinghdbc,serverhdbc;                     //连接句柄
SQLHSTMT    kinghstmt,serverhstmt;                   //语句句柄
SQLRETURN   ret;
SQLCHAR     sName[NAME_LEN],sDepart[DEPART_LEN],
            sSex[SSEX_LEN], sSno[SNO_LEN];
SQLINTEGER  sAge;
SQLINTEGER  cbAge = 0, cbSno = SQL_NTS, cbSex = SQL_NTS,
            cbName = SQL_NTS,cbDepart = SQL_NTS;
```

```
    /* Step 2 初始化环境 */
ret = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE, &kinghenv);
ret = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE, &serverhenv);
ret = SQLSetEnvAttr(kinghenv,SQL_ATTR_ODBC_VERSION, (void *)SQL_OV_ODBC3, 0);
ret = SQLSetEnvAttr(serverhenv,SQL_ATTR_ODBC_VERSION, (void *)SQL_OV_ODBC3, 0);
   /* Step 3 :建立连接 */
ret = SQLAllocHandle(SQL_HANDLE_DBC, kinghenv, &kinghdbc);
ret = SQLAllocHandle(SQL_HANDLE_DBC, serverhenv, &serverhdbc);
ret = SQLConnect(kinghdbc,"KingbaseES ODBC",SQL_NTS, "SYSTEM",
                 SQL_NTS, "MANAGER",SQL_NTS);
if (! SQL_SUCCEEDED(ret))                          //连接失败时返回错误值
      return -1;
ret = SQLConnect(serverhdbc, "SQLServer", SQL_NTS, "sa",
                 SQL_NTS,"sa",SQL_NTS);
if (! SQL_SUCCEEDED(ret) )                         //连接失败时返回错误值
      return -1;
      /* Step 4 :初始化语句句柄 */
ret = SQLAllocHandle(SQL_HANDLE_STMT, kinghdbc, &kinghstmt);
ret = SQLSetStmtAttr(kinghstmt,SQL_ATTR_ROW_BIND_TYPE, (SQLPOINTER)SQL_BIND_BY_
COLUMN,SQL_IS_INTEGER );
ret = SQLAllocHandle(SQL_HANDLE_STMT, serverhdbc, &serverhstmt);
     /* Step 5 :两种方式执行语句 */
     /* 预编译带有参数的语句 */
ret = SQLPrepare (serverhstmt," INSERT INTO STUDENT (SNO, SNAME, SSEX, SAGE, SDEPT)
VALUES (?, ?, ?, ?, ?)", SQL_NTS);
if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO)
{
ret = SQLBindParameter(serverhstmt,1,SQL_PARAM_INPUT,SQL_C_CHAR,
                       SQL_CHAR, SNO_LEN, 0, sSno, 0, &cbSno);
ret = SQLBindParameter(serverhstmt,2,SQL_PARAM_INPUT, SQL_C_CHAR,
                       SQL_CHAR, NAME_LEN, 0, sName, 0, &cbName);
ret = SQLBindParameter(serverhstmt,3,SQL_PARAM_INPUT,SQL_C_CHAR,
                       SQL_CHAR, 2, 0, sSex, 0, &cbSex);
ret = SQLBindParameter(serverhstmt,4,SQL_PARAM_INPUT,
                       SQL_C_LONG,SQL_INTEGER, 0, 0, &sAge, 0, &cbAge);
ret = SQLBindParameter(serverhstmt,5,SQL_PARAM_INPUT,SQL_C_CHAR,
                       SQL_CHAR, DEPART_LEN, 0, sDepart, 0, &cbDepart);
}
   /* 执行 SQL 语句 */
ret = SQLExecDirect(kinghstmt,"SELECT * FROM STUDENT",SQL_NTS);
if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO)
{
ret = SQLBindCol(kinghstmt, 1, SQL_C_CHAR,sSno,SNO_LEN,&cbSno);
```

```
ret = SQLBindCol(kinghstmt, 2, SQL_C_CHAR,sName,NAME_LEN, &cbName);
ret = SQLBindCol(kinghstmt, 3, SQL_C_CHAR, sSex, SSEX_LEN, &cbSex);
ret = SQLBindCol(kinghstmt, 4, SQL_C_LONG, &sAge, 0, &cbAge);
ret = SQLBindCol(kinghstmt, 5, SQL_C_CHAR, sDepart, DEPART_LEN, &cbDepart);
}
  /* Step 6 :处理结果集并执行预编译后的语句 */
while ( (ret = SQLFetch(kinghstmt) ) != SQL_NO_DATA_FOUND)
{
    if(ret == SQL_ERROR) printf("Fetch error\n");
    else ret = SQLExecute(serverhstmt);
}
   /* Step 7 中止处理 */
SQLFreeHandle(SQL_HANDLE_STMT,kinghstmt);
SQLDisconnect(kinghdbc);
SQLFreeHandle(SQL_HANDLE_DBC, kinghdbc);
SQLFreeHandle(SQL_HANDLE_ENV,kinghenv);
SQLFreeHandle(SQL_HANDLE_STMT, serverhstmt);
SQLDisconnect(serverhdbc);
SQLFreeHandle(SQL_HANDLE_DBC, serverhdbc);
SQLFreeHandle(SQL_HANDLE_ENV, serverhenv);
return 0;
}
```

二、初始化环境

由于还没有和具体的驱动程序相关联,不是由具体的数据库管理系统驱动程序来进行管理,而是由 Driver Manager 来进行控制,并配置环境属性。直到应用程序通过调用连接函数和某个数据源进行连接后,Driver Manager 才调用所连的驱动程序中的 SQLAllocHandle,来真正分配环境句柄的数据结构。

三、建立连接

应用程序调用 SQLAllocHandle 分配连接句柄,通过 SQLConnect、SQLDriverConnect 或 SQLBrowseConnect 与数据源连接。其中 SQLConnect 是最简单的连接函数,输入参数为配置好的数据源名称、用户 ID 和口令。本例中 KingbaseES ODBC 为数据源名字,SYSTEM 为用户名,而 MANAGER 为用户密码,需要注意的是使用 KINGBASE 时用户名和密码必须要求大写。

四、分配语句句柄

在处理任何 SQL 语句之前,应用程序还需要首先分配一个语句句柄。语句句柄含有具体的 SQL 语句以及输出的结果集等信息。在后面的执行函数中,语句句柄都是必要的输入参数。本例中分配了两个语句句柄,一个用来从 KingbaseES 中读取数据产生结果集(kinghstmt),另一个用来向 SQLSERVER 插入数据(serverhstmt)。

应用程序还可以通过 SQLtStmtAttr 来设置语句属性(也可以使用默认值),本例中结果

集绑定的方式为按列绑定。

五、执行 SQL 语句

应用程序处理 SQL 语句的方式有两种：预处理（SQLPrepare、SQLExecute 适用于语句的多次执行）或直接执行（SQLExecdirect）。如果 SQL 语句含有参数，应用程序为每个参数调用 SQLBindParameter，并把它们绑定至应用程序变量。这样应用程序可以直接通过改变应用程序缓冲区的内容从而在程序中动态的改变 SQL 语句的具体执行。接下来的操作则会根据语句的类型来进行相应处理。

- 有结果集的语句（select 或是编目函数），则进行结果集处理。
- 没有结果集的函数，可以直接利用本语句句柄继续执行新的语句或是获取行计数（本次执行所影响的行数）之后继续执行。

在本例中，使用 SQLExecdirect 获取 KingbaseES 中的结果集，并将结果集根据各列不同的数据类型绑定到用户程序缓冲区。

在插入数据时，采用了预编译的方式，首先通过 SQLPrepare 来预处理 SQL 语句，将每一列绑定到用户缓冲区。

可以看到在本例中，结果集的用户缓冲区和要插入数据的缓冲区是一致的，这是为了直接使用结果集中的数据，应用程序也可以直接修改缓冲区的内容，插入需要的数据。在这里需要注意的是应用程序还没有真正的执行插入，而只是填好了要插入的数据，真正的执行是在 SQLExecute 中进行的。这也恰恰是预编译的好处，只要进行一次语法分析，就可以多次执行同一语句。

六、结果集处理

应用程序可以通过 SQLNumResultCols 来获取结果集中的列数；通过 SQLDescribeCol 或是 SQLColAttrbute 函数来获取结果集每一列的名称、数据类型、精度和范围。以上两步对于信息明确的函数是可以省略的。

ODBC 中使用游标来处理结果集数据。游标可以分为 forward-only 游标和可滚动（scroll）游标。forward-only 游标只能在结果集中向前滚动，它是 ODBC 的默认游标类型。可滚动（scroll）游标又可以分为静态（static）、动态（dynamic），码集驱动（keyset-driven）和混合型（mixed）四种。

ODBC 游标的打开方式不同于嵌入式 SQL，不是显式声明而是系统自动产生一个游标（Cursor），当结果集刚刚生成时，游标指向第一行数据之前。应用程序通过 SQLBindCol，把查询结果绑定到应用程序缓冲区中，通过 SQLFetch 或是 SQLFetchScroll 来移动游标获取结果集中的每一行数据。对于如图像这类特别的数据类型当一个缓冲区不足以容纳所有的数据时，可以通过 SQLGetdata 分多次获取。最后通过 SQLClosecursor 来关闭游标。在本例中，使用了 SQLFetch 来不断地获取结果集，对于结果集中的每行数据，读取到用户缓冲区，这时使用连接到 SQLSERVER 的语句句柄，执行预编译后的 SQL 语句进行插入，当处理到结果集

的末尾时，该函数返回 SQL_NO_DATA_FOUND，退出循环。

七、中止处理

处理结束后，应用程序将首先释放语句句柄，然后释放数据库连接，并与数据库服务器断开，最后释放 ODBC 环境。

8.3.5 小结

本节阐述了 ODBC 提出的目的是为了提高应用系统与数据库平台的独立性。使用 ODBC 使得应用系统的移植变得容易。当一个应用系统从一个数据库平台移植到另一个数据库平台时只要改换 ODBC 中 RDBMS 驱动程序就行了。

使用 ODBC 可以使得应用系统的开发与数据库平台的选择、数据库设计等工作并行进行。你可以在现有的数据库平台上开发应用系统，然后方便地移植到选择的数据库平台上，从而大大缩短整个系统的开发时间。

实验 5 通过嵌入式 SQL 访问数据库

一、实验目的

熟悉通过嵌入式 SQL(主语言为 C 语言)编程访问数据库

二、实验平台和实验工具

在实验 1 中安装的 RDBMS 及其预编译程序。编程工具自选。如果你安装的是 Kingbase ES，Kingbase ES 的预编译程序是 esglc.exe。esqlc 详细的使用方法请参阅 Kingbase ES 的联机帮助。

三、实验内容和要求

熟悉 RDBMS 的预编译程序。通过嵌入式 SQL 编程访问数据库的基本步骤对学生－课程数据库中的表，完成下面功能：(你也可以自己给出功能要求)

1. 查询某一门课程的信息。要查询的课程由用户在程序运行过程中指定，放在主变量中。

2. 查询选修某一门课程的选课信息，要查询的课程号由用户在程序运行过程中指定，放在主变量中，然后根据用户的要求修改其中某些记录的成绩字段。

要求：提交源程序并标识必要的注释。保证程序能正确编译和运行，认真填写实验报告。

实验 6 使用 PL/SQL 编写存储过程访问数据库

一、实验目的

熟悉使用存储过程来进行数据库应用程序的设计。

二、实验平台和实验工具

实验平台：在实验 1 中安装的 RDBMS 及其 PL/SQL 执行引擎。编程工具自选。

三、实验内容和要求

对学生－课程数据库，编写存储过程，完成下面功能：

1. 统计离散数学的成绩分布情况,即按照各分数段统计人数;

2. 统计任意一门课的平均成绩;

3. 将学生选课成绩从百分制改为等级制(即A、B、C、D、E)。

要求:提交源程序并标识必要的注释。保证程序能正确编译和运行,认真填写实验报告。

实验7　通过ODBC访问数据库

一、实验目的

学会配置ODBC数据源。熟悉使用ODBC来进行数据库应用程序的设计,熟悉通过ODBC接口访问(异构)数据库并对(异构)数据库进行操作。

二、实验平台和实验工具

实验平台:在实验1中安装的RDBMS及其ODBC Driver。

通过C语言编写访问数据库的应用程序。编程工具自选。

三、实验内容和要求

使用ODBC编写应用程序来对异构数据库进行各种数据操作。

配置两个不同的数据源,编写程序连接两个不同RDBMS的数据源,对异构数据库进行操作。例如,将KingbaseES数据库的某个表中的数据转移到SQL Server数据库的表中。

要求:给出配置两个不同的数据源的过程。提交源程序并标识必要的注释。保证程序能正确编译和运行,认真填写实验报告。

*实验8　通过JDBC访问数据库

本章没有讲解JDBC编程的方法,要完成本实验读者可以参考Kingbase ES有关JDBC的使用说明。在书后的光盘中给出了实验报告的示例。供读者自学参考。

一、实验目的

熟悉使用JDBC来进行数据库应用程序的设计,熟悉通过JDBC接口编程访问数据库并对数据库进行操作。

二、实验平台和实验工具

实验平台:在实验1中安装的RDBMS及其JDBC Driver。使用Java语言编写有关应用程序对数据库进行数据操作。编程工具自选。

三、实验内容和要求

使用Java语言编程实现对数据库的访问,数据库建议选用学生-课程数据库,读者可以选择进行创建、插入、查询、删除和更新等操作。

要求:提交源程序并标识必要的注释。保证程序能正确编译和运行,认真填写实验报告。

四、实验报告

《数据库系统概论》实验报告				
题目：	姓名		日期	
实验环境：				
实验内容与完成情况：				
出现的问题：				
解决方案(列出遇到的问题和解决办法,列出没有解决的问题)：				

本章参考文献

1 安装的 RDBMS 用户使用手册。

2 安装的 RDBMS 编程手册。

3 关于 Java 和 JDBC,参阅网站 www.sun.com,在上面可获取各类 Java 信息和最新资料。

4 王珊.数据库系统简明教程.北京:高等教育出版社,2004

第三篇 系 统 篇

本篇讲解数据库管理系统中查询处理和事务管理的基本概念和基础知识。

系统篇包括4章。

第九章关系查询处理和查询优化，讲解查询处理和查询优化的基本概念、方法和技术。

第十章数据库恢复技术，介绍事务的基本概念和事务的性质。讲解数据库系统遇到故障后进行恢复技术和方法。

第十一章并发控制，介绍并发操作可能造成数据不一致的问题，讲解并发控制的基本概念和最常用的封锁技术。

第十二章数据库管理系统，阐述DBMS的基本功能，系统结构及主要的实现技术。本章可以作为选读内容，为读者进一步学习数据库内部实现技术打下基础。

第九章 关系查询处理和查询优化

本章介绍关系数据库的查询优化技术。为此首先介绍 RDBMS 的查询处理步骤，然后介绍查询优化技术。查询优化一般可分为代数优化和物理优化。代数优化是指关系代数表达式的优化；物理优化则是指存取路径和底层操作算法的选择，所以本章讲解了实现查询操作的主要算法。本章的目的是使读者初步了解 RDBMS 查询处理的基本步骤，查询优化的概念、基本方法和技术，为数据库应用开发中利用查询优化技术提高查询效率和系统性能打下基础。

9.1 关系数据库系统的查询处理

查询处理的任务是把用户提交给 RDBMS 的查询语句转换为高效的执行计划。

9.1.1 查询处理步骤

RDBMS 查询处理可以分为 4 个阶段：查询分析、查询检查、查询优化和查询执行，如图 9.1 所示。

1. 查询分析

首先，对查询语句进行扫描、词法分析和语法分析。从查询语句中识别出语言符号，如 SQL 关键字、属性名和关系名等，进行语法检查和语法分析，即判断查询语句是否符合 SQL 语法规则。

2. 查询检查

根据数据字典对合法的查询语句进行语义检查，即检查语句中的数据库对象，如属性名、关系名，是否存在和是否有效。还要根据数据字典中的用户权限和完整性约束定义对用户的存取权限进行检查。如果该用户没有相应的访问权限或违反了完整性约束，就拒绝执行该查询。检查通过后便把 SQL 查询语句转换成等价的关系代数表达式。RDBMS 一般都用**查询树**(**query tree**)，也称为**语法分析树**(**syntax tree**)，来表示扩展的关系代数表达式。这个过程中要把数据库对象的外部名称转换为内部表示。

3. 查询优化

每个查询都会有许多可供选择的执行策略和操作算法，查询优化(query optimization)就是选择一个高效执行的查询处理策略。查询优化有多种方法。按照优化的层次一般可分为代数优化和物理优化。代数优化是指关系代数表达式的优化，即按照一定的规则，改变代数表达式中操作的次序和组合，使查询执行更高效；物理优化则是指存取路径和底层操作算法的选择。选择的依据可以是基于规则(rule based)的，也可以是基于代价(cost based)的，还可以是基于语义(semantic based)的。

实际 RDBMS 中的查询优化器都综合运用了这些优化技术，以获得最好的查询优化效果。

4. 查询执行

依据优化器得到的执行策略生成查询计划，由**代码生成器(code generator)**生成执行这个查询计划的代码。

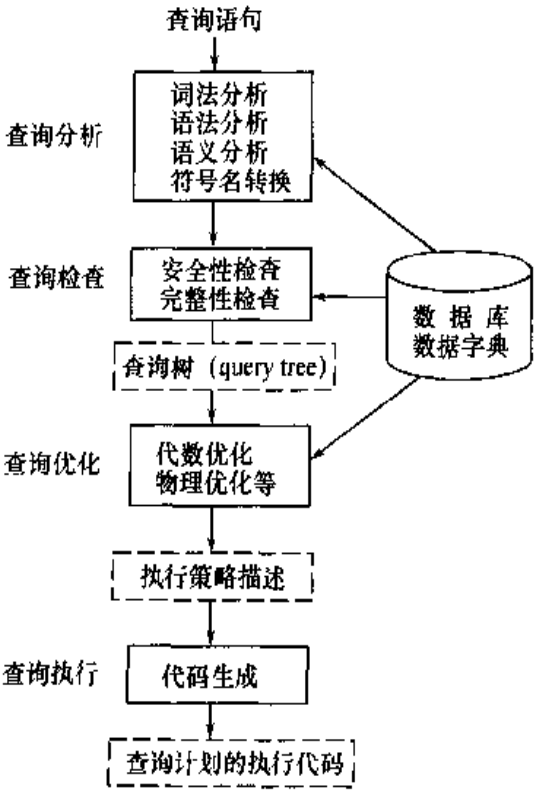


图 9.1 查询处理步骤

9.1.2 实现查询操作的算法示例

本节简单介绍选择操作和连接操作的实现算法，确切地说是算法思想。每一种操作有多种执行这个操作的算法，这里仅仅介绍最主要的几个算法。对于其他重要操作的详细实现算法，有兴趣的读者请参考有关 RDBMS 实现技术的书。

一、选择操作的实现

第三章中已经讲解了 SELECT 语句功能十分强大，有许多选项，因此实现的算法和优化策略也很复杂。不失一般性，下面以简单的选择操作为例介绍典型的实现方法。

[例 1] Select * from student where <条件表达式>；

考虑<条件表达式>的几种情况：

C1：无条件；
C2：Sno ='200215121'；
C3：Sage > 20；
C4：Sdept ='CS' AND Sage > 20；

1. 简单的全表扫描方法

对查询的基本表顺序扫描，逐一检查每个元组是否满足选择条件，把满足条件的元组作

为结果输出。对于小表,这种方法简单有效。对于大表顺序扫描十分费时,效率很低。

2. 索引(或散列)扫描方法

如果选择条件中的属性上有索引(例如 B + 树索引或 Hash 索引),可以用索引扫描方法。通过索引先找到满足条件的元组主码或元组指针,再通过元组指针直接在查询的基本表中找到元组。

[**例 1 - C2**] 以 C2 为例,Sno = '200215121',并且 Sno 上有索引(或 Sno 是散列码),则可以使用索引(或散列)得到 Sno 为'200215121'元组的指针,然后通过元组指针在 student 表中检索到该学生。

[**例 1 - C3**] 以 C3 为例,Sage > 20,并且 Sage 上有 B + 树索引,则可以使用 B + 树索引找到 Sage = 20 的索引项,以此为入口点在 B + 树的顺序集上得到 Sage > 20 的所有元组指针,然后通过这些元组指针到 student 表中检索到所有年龄大于 20 的学生。

[**例 1 - C4**] 以 C4 为例,Sdept = 'CS' AND Sage > 20,如果 Sdept 和 Sage 上都有索引,一种算法是:分别用上面两种方法分别找到 Sdept = 'CS'的一组元组指针和 Sage > 20 的另一组元组指针,求这 2 组指针的交集,再到 student 表中检索,就得到计算机系年龄大于 20 的学生。

另一种算法是:找到 Sdept = 'CS'的一组元组指针,通过这些元组指针到 student 表中检索,并对得到的元组检查另一些选择条件(如 Sage > 20)是否满足,把满足条件的元组作为结果输出。

二、连接操作的实现

连接操作是查询处理中最耗时的操作之一。不失一般性,这里只讨论等值连接(或自然连接)最常用的实现算法。

[**例 2**] SELECT * FROM Student,SC WHERE Student. Sno = SC. Sno;

1. 嵌套循环方法(nested loop)

这是最简单可行的算法。对外层循环(Student)的每一个元组(s),检索内层循环(SC)中的每一个元组(sc),并检查这两个元组在连接属性(sno)上是否相等。如果满足连接条件,则串接后作为结果输出,直到外层循环表中的元组处理完为止。

2. 排序 - 合并方法(sort-merge join 或 merge join)

这也是常用的算法,尤其适合连接的诸表已经排好序的情况。

用排序-合并连接方法的步骤是:

① 如果连接的表没有排好序,首先对 Student 表和 SC 表按连接属性 Sno 排序;

② 取 Student 表中第一个 Sno,依次扫描 SC 表中具有相同 Sno 的元组;把它们连接起来(如图 9.2 所示);

③ 当扫描到 Sno 不相同的第一个 SC 元组时,返回 Student 表扫描它的下一个元组,再扫描 SC 表中具有相同 Sno 的元组,把它们连接起来。

重复上述步骤直到 Student 表扫描完。

这样 Student 表和 SC 表都只要扫描一遍。当然，如果 2 个表原来无序，执行时间要加上对两个表的排序时间。即使这样，对于 2 个大表，先排序后使用 sort-merge join 方法执行连接，总的时间一般仍会大大减少。

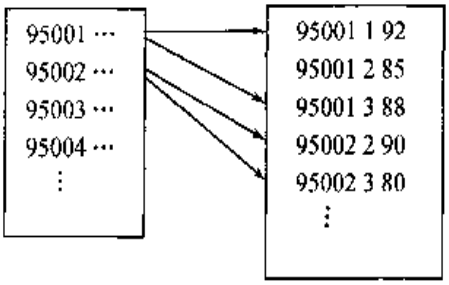


图 9.2 排序－合并连接方法示意图

3. 索引连接(index join)方法

用索引连接方法的步骤是：

① 在 SC 表上建立属性 Sno 的索引，如果原来没有的话；

② 对 Student 中每一个元组，由 Sno 值通过 SC 的索引查找相应的 SC 元组；

③ 把这些 SC 元组和 Student 元组连接起来。

循环执行②③，直到 Student 表中的元组处理完为止。

4. Hash Join 方法

把连接属性作为 hash 码，用同一个 hash 函数把 R 和 S 中的元组散列到同一个 hash 文件中。第一步，划分阶段(partitioning phase)，对包含较少元组的表(比如 R)进行一遍处理，把它的元组按 hash 函数分散到 hash 表的桶中；第二步，试探阶段(probing phase)，也称为连接阶段(join phase)，对另一个表(S)进行一遍处理，把 S 的元组散列到适当的 hash 桶中，并把元组与桶中所有来自 R 并与之相匹配的元组连接起来。

上面 hash join 算法假设两个表中较小的表在第一阶段后可以完全放入内存的 hash 桶中。不需要这个前提条件的 hash join 算法以及许多改进的算法这里就不介绍了。以上的算法思想可以推广到更加一般的多个表的连接算法上。请读者自己完成。

9.2 关系数据库系统的查询优化

查询优化在关系数据库系统中有着非常重要的地位。关系数据库系统和非过程化的 SQL 之所以能够取得巨大的成功，关键是得益于查询优化技术的发展。关系查询优化是影响 RDBMS 性能的关键因素。

优化对关系系统来说既是挑战又是机遇。所谓挑战是指关系系统为了达到用户可接受的性能必须进行查询优化。由于关系表达式的语义级别很高，使关系系统可以从关系表达式中分析查询语义，提供了执行查询优化的可能性。这就为关系系统在性能上接近甚至超过非关系系统提供了机遇。

9.2.1 查询优化概述

关系系统的查询优化既是 RDBMS 实现的关键技术又是关系系统的优点所在。它减轻

了用户选择存取路径的负担。用户只要提出“干什么”,不必指出“怎么干”。对比一下非关系系统中的情况:用户使用过程化的语言表达查询要求,执行何种记录级的操作,以及操作的序列是由用户而不是由系统来决定的。因此用户必须了解存取路径,系统要提供用户选择存取路径的手段,查询效率由用户的存取策略决定。如果用户做了不当的选择,系统是无法对此加以改进的。这就要求用户有较高的数据库技术和程序设计水平。

查询优化的优点不仅在于用户不必考虑如何最好地表达查询以获得较好的效率,而且在于系统可以比用户程序的“优化”做得更好。这是因为:

(1) 优化器可以从数据字典中获取许多统计信息,例如每个关系表中的元组数、关系中每个属性值的分布情况、哪些属性上已经建立了索引等。优化器可以根据这些信息做出正确的估算,选择高效的执行计划,而用户程序则难以获得这些信息。

(2) 如果数据库的物理统计信息改变了,系统可以自动对查询进行重新优化以选择相适应的执行计划。在非关系系统中必须重写程序,而重写程序在实际应用中往往是不太可能的。

(3) 优化器可以考虑数百种不同的执行计划,而程序员一般只能考虑有限的几种可能性。

(4) 优化器中包括了很多复杂的优化技术,这些优化技术往往只有最好的程序员才能掌握。系统的自动优化相当于使得所有人都拥有这些优化技术。目前 RDBMS 通过某种代价模型计算出各种查询执行策略的执行代价,然后选取代价最小的执行方案。在集中式数据库中,查询的执行开销主要包括磁盘存取块数(I/O 代价),处理机时间(CPU 代价),查询的内存开销。在分布式数据库中还要加上通信代价,即:

总代价 = I/O 代价 + CPU 代价 + 内存代价 + 通信代价

一般地,集中式数据库中 I/O 代价是最主要的。

查询优化的总目标是:选择有效的策略,求得给定关系表达式的值,使得查询代价最小(实际上是较小)。

9.2.2 一个实例

首先来看一个简单的例子,说明为什么要进行查询优化。

[例 3] 求选修了 2 号课程的学生姓名。用 SQL 表达:

```
SELECT Student.Sname
FROM Student,SC
WHERE Student.Sno = SC.Sno AND SC.Cno = '2';
```

假定学生 - 课程数据库中有 1000 个学生记录,10000 个选课记录,其中选修 2 号课程的选课记录为 50 个。

系统可以用多种等价的关系代数表达式来完成这一查询

$$Q_1 = \pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge Sc_Cno='2'}(Student \times SC))$$
$$Q_2 = \pi_{Sname}(\sigma_{Sc_Cno='2'}(Student \bowtie SC))$$
$$Q_3 = \pi_{Sname}(Student \bowtie \sigma_{Sc_Cno='2'}(SC))$$

还可以写出几种等价的关系代数表达式,但分析这三种就足以说明问题了。后面将看到由于查询执行的策略不同,查询时间相差很大。

一、第一种情况

1. 计算广义笛卡尔积

把 Student 和 SC 的每个元组连接起来。一般连接的做法是:在内存中尽可能多地装入某个表(如 Student 表)的若干块,留出一块存放另一个表(如 SC 表)的元组。然后把 SC 中的每个元组和 Student 中每个元组连接,连接后的元组装满一块后就写到中间文件上,再从 SC 中读入一块和内存中的 Student 元组连接,直到 SC 表处理完。这时再一次读入若干块 Student 元组,读入一块 SC 元组,重复上述处理过程,直到把 Student 表处理完。

设一个块能装 10 个 Student 元组或 100 个 SC 元组,在内存中存放 5 块 Student 元组和 1 块 SC 元组,则读取总块数为

$$\frac{1\,000}{10} + \frac{1\,000}{10 \times 5} \times \frac{10\,000}{100} = 100 + 20 \times 100 = 2\,100 \text{ 块}$$

其中,读 Student 表 100 块。读 SC 表 20 遍,每遍 100 块。若每秒读写 20 块,则总计要花 105 s。

连接后的元组数为 $10^3 \times 10^4 = 10^7$。设每块能装 10 个元组,则写出这些块要用 $10^6/20 = 5 \times 10^4$ s。

2. 作选择操作

依次读入连接后的元组,按照选择条件选取满足要求的记录。假定内存处理时间忽略。这一步读取中间文件花费的时间(同写中间文件一样)需 5×10^4 s。满足条件的元组假设仅 50 个,均可放在内存。

3. 作投影操作

把第 2 步的结果在 Sname 上作投影输出,得到最终结果。

因此第一种情况下执行查询的总时间 $\approx 105 + 2 \times 5 \times 10^4 \approx \mathbf{10^5}$ s。这里,所有内存处理时间均忽略不计。

二、第二种情况

1. 计算自然连接

为了执行自然连接,读取 Student 和 SC 表的策略不变,总的读取块数仍为 2 100 块花费 105 s。但自然连接的结果比第一种情况大大减少,为 10^4 个。因此写出这些元组时间为 $10^4/10/20 = 50$ s,仅为第一种情况的千分之一。

2. 读取中间文件块,执行选择运算,花费时间也为 50 s。

3. 把第 2 步结果投影输出。

第二种情况总的执行时间≈105 + 50 + 50≈**205 s**。

三、第三种情况

1. 先对 SC 表作选择运算，只需读一遍 SC 表，存取 100 块花费时间为 5 s，因为满足条件的元组仅 50 个，不必使用中间文件。

2. 读取 Student 表，把读入的 Student 元组和内存中的 SC 元组作连接。也只需读一遍 Student 表共 100 块，花费时间为 **5 s**。

3. 把连接结果投影输出。

第三种情况总的执行时间≈5 + 5≈**10 s**。

假如 SC 表的 Cno 字段上有索引，第一步就不必读取所有的 SC 元组而只需读取 Cno = '2'的那些元组(50 个)。存取的索引块和 SC 中满足条件的数据块大约总共 3 ~ 4 块。若 Student 表在 Sno 上也有索引，则第二步也不必读取所有的 Student 元组，因为满足条件的 SC 记录仅 50 个，涉及最多 50 个 Student 记录，因此读取 Student 表的块数也可大大减少。总的存取时间将进一步减少到**数秒**。

这个简单的例子充分说明了查询优化的必要性，同时也给出一些查询优化方法的初步概念。把代数表达式 Q_1 变换为 Q_2、Q_3，即有选择和连接操作时，应当先做选择操作，这样参加连接的元组就可以大大减少，这是代数优化。在 Q_3 中，SC 表的选择操作算法有全表扫描和索引扫描 2 种方法，经过初步估算，索引扫描方法较优。同样对于 Student 和 SC 表的连接，利用 Student 表上的索引，采用 index join 代价也较小，这就是物理优化。

9.3 代数优化

9.1 中已经讲解了 SQL 语句经过查询分析、查询检查后变换为查询树，它是关系代数表达式的内部表示。本节介绍基于关系代数等价变换规则的优化方法，即代数优化。

9.3.1 关系代数表达式等价变换规则

代数优化策略是通过对关系代数表达式的等价变换来提高查询效率。所谓关系代数表达式的等价是指用相同的关系代替两个表达式中相应的关系所得到的结果是相同的。两个关系表达式 E_1 和 E_2 是等价的，可记为 $E_1 \equiv E_2$。

下面是常用的等价变换规则，证明从略

1. 连接、笛卡尔积交换律

设 E_1 和 E_2 是关系代数表达式，F 是连接运算的条件，则有

$$E_1 \times E_2 \equiv E_2 \times E_1$$

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$

$$E_1 \underset{F}{\bowtie} E_2 \equiv E_2 \underset{F}{\bowtie} E_1$$

2. 连接、笛卡尔积的结合律

设 E_1,E_2,E_3 是关系代数表达式,F_1 和 F_2 是连接运算的条件,则有

$$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3)$$

$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$$

$$(E_1 \underset{F_1}{\bowtie} E_2) \underset{F_2}{\bowtie} E_3 \equiv E_1 \underset{F_1}{\bowtie} (E_2 \underset{F_2}{\bowtie} E_3)$$

3. 投影的串接定律

$$\pi_{A_1,A_2,\cdots,A_n}(\pi_{B_1,B_2,\cdots,B_m}(E)) \equiv \pi_{A_1,A_2,\cdots,A_n}(E)$$

这里,E 是关系代数表达式,$A_i(i=1,2,\cdots,n)$,$B_j(j=1,2,\cdots,m)$ 是属性名且 $\{A_1,A_2,\cdots,A_n\}$ 构成 $\{B_1,B_2,\cdots,B_m\}$ 的子集。

4. 选择的串接定律

$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E)$$

这里,E 是关系代数表达式,F_1、F_2 是选择条件。选择的串接律说明选择条件可以合并。这样一次就可检查全部条件。

5. 选择与投影操作的交换律

$$\sigma_F(\pi_{A_1,A_2,\cdots,A_n}(E)) \equiv \pi_{A_1,A_2,\cdots,A_n}(\sigma_F(E))$$

这里,选择条件 F 只涉及属性 $A_1,\cdots,A_n$。

若 F 中有不属于 $A_1,\cdots,A_n$ 的属性 $B_1,\cdots,B_m$ 则有更一般的规则:

$$\pi_{A_1,A_2,\cdots,A_n}(\sigma_F(E)) \equiv \pi_{A_1,A_2,\cdots,A_n}(\sigma_F(\pi_{A_1,A_2,\cdots,A_n,B_1,B_2,\cdots,B_m}(E)))$$

6. 选择与笛卡尔积的交换律

如果 F 中涉及的属性都是 E_1 中的属性,则

$$\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2$$

如果 $F = F_1 \wedge F_2$,并且 F_1 只涉及 E_1 中的属性,F_2 只涉及 E_2 中的属性,则由上面的等价变换规则 1,4,6 可推出:

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_1}(E_1) \times \sigma_{F_2}(E_2)$$

若 F_1 只涉及 E_1 中的属性,F_2 涉及 E_1 和 E_2 两者的属性,则仍有

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_2}(\sigma_{F_1}(E_1) \times E_2)$$

它使部分选择在笛卡尔积前先做。

7. 选择与并的分配律

设 $E = E_1 \cup E_2$,E_1,E_2 有相同的属性名,则

$$\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2)$$

8. 选择与差运算的分配律

若 E_1 与 E_2 有相同的属性名,则

$$\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$$

9. 选择对自然连接的分配律

$$\sigma_F(E_1 \bowtie E_2) \equiv \sigma_F(E_1) \bowtie \sigma_F(E_2)$$

F 只涉及 E_1 与 E_2 的公共属性。

10. 投影与笛卡尔积的分配律

设 E_1 和 E_2 是两个关系表达式，$A_1,\cdots,A_n$ 是 E_1 的属性，$B_1,\cdots,B_m$ 是 E_2 的属性，则

$$\pi_{A_1,A_2,\cdots,A_n,B_1,B_2,\cdots,B_m}(E_1 \times E_2) \equiv \pi_{A_1,A_2,\cdots,A_n}(E_1) \times \pi_{B_1,B_2,\cdots,B_m}(E_2)$$

11. 投影与并的分配律

设 E_1 和 E_2 有相同的属性名，则

$$\pi_{A_1,A_2,\cdots,A_n}(E_1 \cup E_2) \equiv \pi_{A_1,A_2,\cdots,A_n}(E_1) \cup \pi_{A_1,A_2,\cdots,A_n}(E_2)$$

9.3.2 查询树的启发式优化

本节讨论应用启发式规则(heuristic rules)的代数优化。这是对关系代数表达式的查询树进行优化的方法。典型的启发式规则有：

1. 选择运算应尽可能先做。在优化策略中这是最重要、最基本的一条。它常常可使执行时节约几个数量级，因为选择运算一般使计算的中间结果大大变小。

2. 把投影运算和选择运算同时进行。如有若干投影和选择运算，并且它们都对同一个关系操作，则可以在扫描此关系的同时完成所有的这些运算以避免重复扫描关系。

3. 把投影同其前或其后的双目运算结合起来，没有必要为了去掉某些字段而扫描一遍关系。

4. 把某些选择同在它前面要执行的笛卡尔积结合起来成为一个连接运算，连接特别是等值连接运算要比同样关系上的笛卡尔积省很多时间。

5. 找出公共子表达式。如果这种重复出现的子表达式的结果不是很大的关系，并且从外存中读入这个关系比计算该子表达式的时间少得多，则先计算一次公共子表达式并把结果写入中间文件是合算的。当查询的是视图时，定义视图的表达式就是公共子表达式的情况。

下面给出遵循这些启发式规则，应用 9.3.1 的等价变换公式来优化关系表达式的算法。

算法：关系表达式的优化。

输入：一个关系表达式的查询树。

输出：优化的查询树。

方法：

(1) 利用等价变换规则 4 把形如 $\sigma_{F_1 \wedge F_2 \wedge \cdots \wedge F_n}(E)$ 变换为 $\sigma_{F_1}(\sigma_{F_2}(\cdots(\sigma_{F_n}(E))\cdots))$。

(2) 对每一个选择，利用等价变换规则 4～9 尽可能把它移到树的叶端。

(3) 对每一个投影利用等价变换规则 3,5,10,11 中的一般形式尽可能把它移向树的叶端。

注意:等价变换规则3使一些投影消失,而规则5把一个投影分裂为两个,其中一个有可能被移向树的叶端。

(4) 利用等价变换规则3~5把选择和投影的串接合并成单个选择、单个投影或一个选择后跟一个投影。使多个选择或投影能同时执行,或在一次扫描中全部完成,尽管这种变换似乎违背"投影尽可能早做"的原则,但这样做效率更高。

(5) 把上述得到的语法树的内节点分组。每一双目运算(×,⋈,∪,-)和它所有的直接祖先为一组(这些直接祖先是(σ,π运算)。如果其后代直到叶子全是单目运算,则也将它们并入该组,但当双目运算是笛卡尔积(×),而且后面不是与它组成等值连接的选择时,则不能把选择与这个双目运算组成同一组。把这些单目运算单独分为一组。

[**例4**] 下面给出[**例3**]中SQL语句的代数优化示例。

(1) 把SQL语句转换成查询树,如图9.3所示。

为了使用关系代数表达式的优化法,不妨假设内部表示是关系代数语法树,则上面的查询树如图9.4所示。

(2) 对查询树进行优化。

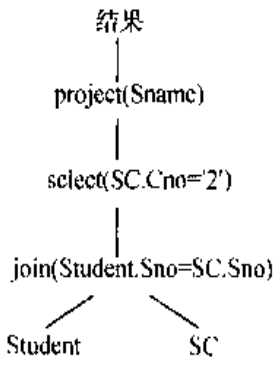


图9.3 查询树

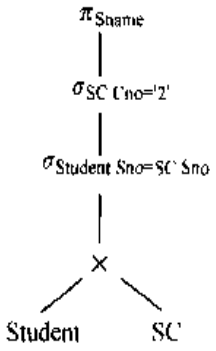


图9.4 关系代数语法树

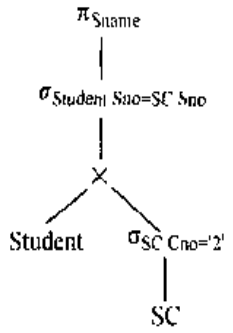


图9.5 优化后的查询树

利用规则4、6把选择$\sigma_{SC.Cno='2'}$移到叶端,图9.4查询树便转换成图9.5优化的查询树。这就是9.2.2节中Q_3的查询树表示。上面已经分析了Q_3比Q_1、Q_2查询效率要高得多。

9.4 物理优化

代数优化改变查询语句中操作的次序和组合,不涉及底层的存取路径。9.1.2小节中已经讲解了对每一种操作有多种执行这个操作的算法,有多条存取路径。因此对于一个查询语句有许多存取方案、它们的执行效率不同,有的会相差很大。因此,仅仅进行代数优化是不够的。物理优化就是要选择高效合理的操作算法或存取路径,求得优化的查询计划,达到查询优化的目标。

选择的方法可以是:

• 基于规则的启发式优化。启发式规则是指那些在大多数情况下都适用,但不是在每种情况下都是适用的规则。

• 基于代价估算的优化。优化器估算不同执行策略的代价,并选出具有最小代价的执行计划。

• 两者结合的优化方法。查询优化器通常会把这两种技术结合在一起使用。因为可能的执行策略很多,要穷尽所有的策略进行代价估算往往是不可行的,会造成查询优化本身付出的代价大于获得的益处。为此,常常先使用启发式规则,选取若干较优的候选方案,减少代价估算的工作量;然后分别计算这些候选方案的执行代价,较快地选出最终的优化方案。

9.4.1 基于启发式规则的存取路径选择优化

一、选择操作的启发式规则有:

1. 对于小关系,使用全表顺序扫描,即使选择列上有索引。

对于大关系,启发式规则有:

2. 对于选择条件是主码 = 值的查询,查询结果最多是一个元组,可以选择主码索引。一般的 RDBMS 会自动建立主码索引。

3. 对于选择条件是非主属性 = 值的查询,并且选择列上有索引,则要估算查询结果的元组数目,如果比例较小(<10%)可以使用索引扫描方法,否则还是使用全表顺序扫描。

4. 对于选择条件是属性上的非等值查询或者范围查询,并且选择列上有索引,同样要估算查询结果的元组数目,如果比例较小(<10%)可以使用索引扫描方法,否则还是使用全表顺序扫描。

5. 对于用 AND 连接的合取选择条件,如果有涉及这些属性的组合索引,则优先采用组合索引扫描方法;如果某些属性上有一般的索引,则可以用[**例 1 - C4**]中介绍的索引扫描方法,否则使用全表顺序扫描。

6. 对于用 OR 连接的析取选择条件,一般使用全表顺序扫描。

二、连接操作的启发式规则有:

1. 如果 2 个表都已经按照连接属性排序,则选用排序 - 合并方法。

2. 如果一个表在连接属性上有索引,则可以选用索引连接方法。

3. 如果上面 2 个规则都不适用,其中一个表较小,则可以选用 Hash join 方法。

4. 最后可以选用嵌套循环方法,并选择其中较小的表,确切地讲是占用的块数(b)较少的表,作为外表(外循环的表)。理由如下:

设连接表 R 与 S 分别占用的块数为 Br 与 Bs,连接操作使用的内存缓冲区块数为 K,分配 K - 1 块给外表。如果 R 为外表,则嵌套循环法存取的块数为 Br + (Br/K - 1)Bs,显然应该选块数小的表作为外表。

上面列出了一些主要的启发式规则，在实际的 RDBMS 中启发式规则要多得多。

9.4.2 基于代价的优化

启发式规则优化是定性的选择，比较粗糙，但是实现简单而且优化本身的代价较小，适合解释执行的系统。因为解释执行的系统，优化开销包含在查询总开销之中。在编译执行的系统中，一次编译优化，多次执行，查询优化和查询执行是分开的。因此，可以采用精细复杂一些的基于代价的优化方法。

一、统计信息

基于代价的优化方法要计算各种操作算法的执行代价，它与数据库的状态密切相关。为此在数据字典中存储了优化器需要的统计信息（database statistics），主要包括如下几个方面。

1. 对每个基本表，该表的元组总数（N）、元组长度（l）、占用的块数（B）、占用的溢出块数（BO）；

2. 对基表的每个列，该列不同值的个数（m）、选择率（f）（如果不同值的分布是均匀的，f = 1/m；如果不同值的分布不均匀，则每个值的选择率 = 具有该值的元组数/N）、该列最大值、最小值，该列上是否已经建立了索引，是哪种索引（B + 树索引、Hash 索引、聚集索引）；

3. 对索引，例如 B + 树索引，该索引的层数（L）、不同索引值的个数、索引的选择基数 S（有 S 个元组具有某个索引值）、索引的叶结点数（Y）；

等等。

二、代价估算示例

下面给出若干操作算法的执行代价估算。

1. 全表扫描算法的代价估算公式

如果基本表大小为 B 块，全表扫描算法的代价 cost = B；

如果选择条件是码 = 值，那么平均搜索代价 cost = B/2。

2. 索引扫描算法的代价估算公式

如果选择条件是码 = 值，如[**例 1 - C2**]，则采用该表的主索引，若为 B + 树，层数为 L，需要存取 B + 树中从根结点到叶结点 L 块，再加上基本表中该元组所在的那一块，所以 cost = L + 1。

如果选择条件涉及非码属性，如[**例 1 - C3**]，若为 B + 树索引，选择条件是相等比较，S 是索引的选择基数（有 S 个元组满足条件）。因为满足条件的元组可能会保存在不同的块上，所以（最坏的情况）cost = L + S。

如果比较条件是 >，>=，<，<= 操作，假设有一半的元组满足条件，那么就要存取一半的叶结点，并通过索引访问一半的表存储块。所以 cost = L + Y/2 + B/2。如果可以获得更准确的选择基数，可以进一步修正 Y/2 与 B/2。

3. 嵌套循环连接算法的代价估算公式

9.4.1 中已经讨论过了嵌套循环连接算法的代价 cost = Br + Br Bs/(K - 1)。如果需要

把连接结果写回磁盘，则 cost = Br + Br Bs/(K - 1) + (Frs * Nr * Ns)/Mrs。其中 Frs 为连接选择性(join selectivity)，表示连接结果元组数的比例，Mrs 是存放连接结果的块因子，表示每块中可以存放的结果元组数目。

4. 排序 - 合并连接算法的代价估算公式

如果连接表已经按照连接属性排好序，则 cost = Br + Bs + (Frs * Nr * Ns)/Mrs。

如果必须对文件排序，那么还需要在代价函数中加上排序的代价。对于包含 B 个块的文件排序的代价大约是$(2 * B) + (2 * B * \log_2 B)$。

上面仅仅列出了少数操作算法的代价估算示例。在实际的 RDBMS 中代价估算公式要多得多，也复杂得多。

9.5 小 结

查询处理是 RDBMS 的核心，而查询优化技术又是查询处理的关键技术。本章仅关注查询(Query)语句，它是 RDBMS 语言处理中最重要、最复杂的部分。更一般的数据库语言(包括 DDL、DML、DCL)处理技术在 RDBMS 实现的书中讲解，本书第十一章也将给出一个概要的介绍。

本章讲解了启发式代数优化、基于规则的存取路径优化和基于代价的优化等方法，实际系统的优化是综合的，优化器是十分复杂的。

本章不是要求读者掌握 RDBMS 查询处理和查询优化的**内部实现**技术，因此没有详细讲解技术细节。本章的目的是希望读者掌握查询优化方法的概念和技术；通过本章实验，进一步了解具体的查询计划表示，能够利用它分析查询的实际执行方案和查询代价，进而通过建立索引或者修改 SQL 语句来降低查询代价，达到优化系统性能的目标。

对于比较复杂的查询，尤其是涉及连接和嵌套的查询，不要把优化的任务全部放在 RDBMS 上，应该找出 RDBMS 的优化规律，以写出适合 RDBMS 自动优化的 SQL 语句。对于 RDBMS 不能优化的查询需要重写查询语句，进行手工调整以优化性能。

习 题

1. 试述查询优化在关系数据库系统中的重要性和可能性。
2. 对学生 - 课程数据库有如下的查询：

```
SELECT Cname
FROM Student, Course, SC
WHERE Student. Sno = SC. Sno AND
      SC. Cno = Course. Cno AND
      Student. Sdept = 'IS';
```

此查询要求信息系学生选修了的所有课程名称。

试画出用关系代数表示的语法树，并用关系代数表达式优化算法对原始的语法树进行优化处理，画出优化后的标准语法树。

3. 试述 RDBMS 查询优化的一般准则。

4. 试述 RDBMS 查询优化的一般步骤。

实验9 查询优化

一、实验目的

1. 了解数据库查询优化方法和查询计划的概念。

2. 学会分析查询的代价，并通过建立索引或者修改 SQL 语句来降低查询代价。

二、实验工具

在实验 1 中安装的 RDBMS 及其查询分析器

三、实验内容和要求

考虑以下 3 种 SQL 操作，查看查询分析器给出的查询计划，分析优化效果。

查询优化可以考虑以下方法：

1）建立索引；

2）重写 SQL 语句（即查询重写）；

3）其他优化方法（调整参数，建立视图或临时表等）。

1. 单表查询 例如以下的查询：（可以自己给出查询语句）

```
select *
from student
where age > 20 ;
```

2. 连接查询，普通的两表连接查询或多表连接查询，分析连接的顺序对查询计划的生成是否有影响？如果有，有何影响？

3. 嵌套查询，自己写几个带有子查询的例子，主要考虑带有 IN 和 EXISTS 谓词的子查询，分为相关子查询和不相关子查询讨论。也可以使用《数据库系统概论》书上列举的例子。

对以上各种查询，比较不同查询计划执行的性能差异，并对实验结果做出分析。

注意：数据库基本表应该有足够大的数据量，才能体现优化的效果。读者可以通过小组的形式就以上的某一方面进行实验、分析和学习。对各自的结果进行讨论。分析和讨论力求做到对 RDBMS 的查询计划和优化方法有较为深刻的了解知识。

说明：在教学网站 http://www.chinadb.org 或 http://4A.hep.com.cn 上给出了本实验的详细要求和实验报告示例，供读者学习参考。

本章参考文献

1 Codd E F. Relational Database: A Practical Foundation for Productivity. CACM 25, No. 2, 1982

（本文是 Codd E F 获得图灵奖后的演说，阐述了关系数据库系统能极大地提高用户生产率这一重要观点。）

2 Schmidt Joachim W, Brodie Michael L (eds.). Relational Database Systems: Analysis and Comparison. New York: Springer-Verlag, 1983

3 Special Issue on Query Optimization. IEEE Database Engineering 5. No. 3, 1982

（[3]是关于查询优化的专辑。）

4 Smith J M, Chang P Y T. Optimizing the Performance of a Relational Algebra Database Interface. CACM 18, No. 10, 1975 （本文研究了关系代数的优化，给出了详细的算法）

5 Wong E, Youssefi K. Decomposition: A Strategy for Query Processing. ACM TODS 1, NO. 3, 1976

（[5]和[6]介绍了 INGRES 采用的查询分解优化方法。）

6 Youssefi X, Wong E. Query Processing in a Relational Database Management System. Proc. 5th International Conference on Very Large Data Bases, 1979

7 Youssefi K, Wong E. Query Processing in a Relational Database Management System. in Proceedings of VLDB, 1979

（本文讨论了 INGRES 查询分解优化算法的性能。）

8 Aho A V, Sagiv Y, Ullman J D. Efficient Optimization of a Class of Relational Expressions. ACM TODS 4, No. 4, 1979 （本文讨论了关系表达式的优化方法。）

9 Astrahan M et al. System R: A Relational Approach to Data Base Management. ACM TODS, 1: 2, 1976 （本文介绍了 SYSTEM R 的基于代价估算的查询优化算法。）

10 Selinger P, et al. Access Path Selection in a Relational Database Management System. in Proceedings of ACM SIGMOD, 1979 （本文讨论了 SYSTEM R 中多表连接操作的查询优化问题。）

11 Astrahan M M, Schkolnick M. Performance of the System R Access Path Selection Mechanism. IFIP, 1980

（本文介绍了 SYSTEM R 的存取路径选择机制及其性能。）

12 Kim W. On Optimizing An SQL-like Nested Query. TODS, 3: 3, September 1982

（本文提出了 SQL 嵌套查询的优化方法。）

13 DeWitt D et al. Implementation Techniques for Main Memory Databases. in Proceedings of ACM SIGMOD. 1984 （本文研究了主存数据库上的查询优化问题。）

14 Yao S B. Optimization of Query Evaluation Algorithms. ACM TODS, 4: 2, 1979

（本文对多个查询优化算法进行了比较和分析。）

15 Jarke M, Koch J. Query Optimization in Database Systems. ACM Computing Surveys, 16: 2, 1984

（本文系统地综述了查询优化的主要研究成果，并列出了这一研究领域的主要文献。）

16 King J. QUIST: A System for Semantic Query Optimization in Relational Databases. In Proceedings of VLDB, 1981 （本文讨论了在数据库中利用语义知识进行查询优化的问题。）

17 Malley C, Zdonick S. A Knowledge-Based Approach to Query Optimization. in Proceedings of the 1st Inter. Conference on Expert Database Systems, 1986

（本文也是讨论在数据库中利用语义知识进行查询优化的问题。）

18 Beck H, Gala S, Navathe S. Classification as a Query Processing Technique in the CANDIDE Semantic Data Model. in Proceedings of IEEE Inter. Conference on Data Engineering, 1989

（本文讨论了一种实现查询优化的分类技术，这种技术适用于基于语义数据模型的数据库系统。）

第十章 数据库恢复技术

本书第十章、第十一章讨论事务(Transaction)处理技术。事务是一系列的数据库操作，是数据库应用程序的基本逻辑单元。事务处理技术主要包括数据库恢复技术和并发控制技术。数据库恢复机制和并发控制机制是数据库管理系统的重要组成部分。本章讨论数据库恢复的概念和常用技术。

10.1 事务的基本概念

在讨论数据库恢复技术之前先讲解事务的基本概念和事务的性质。

一、事务

所谓**事务**是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位。例如，在关系数据库中，一个事务可以是一条 SQL 语句、一组 SQL 语句或整个程序。

事务和程序是两个概念。一般地讲，一个程序中包含多个事务。

事务的开始与结束可以由用户显式控制。如果用户没有显式地定义事务，则由 DBMS 按缺省规定自动划分事务。在 SQL 中，定义事务的语句有 3 条：

```
BEGIN TRANSACTION
COMMIT
ROLLBACK
```

事务通常是以 BEGIN TRANSACTION 开始，以 COMMIT 或 ROLLBACK 结束。COMMIT 表示提交，即提交事务的所有操作。具体地说就是将事务中所有对数据库的更新写回到磁盘上的物理数据库中去，事务正常结束。ROLLBACK 表示回滚，即在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销，回滚到事务开始时的状态。这里的操作指对数据库的更新操作。

二、事务的特性

事务具有四个特性：原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)和持续

性(Durability)。这个四个特性简称为 ACID 特性。

1. 原子性

事务是数据库的逻辑工作单位,事务中包括的诸操作要么都做,要么都不做。

2. 一致性

事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。因此当数据库只包含成功事务提交的结果时,就说数据库处于一致性状态。如果数据库系统运行中发生故障,有些事务尚未完成就被迫中断,这些未完成事务对数据库所做的修改有一部分已写入物理数据库,这时数据库就处于一种不正确的状态,或者说是不一致的状态。例如某公司在银行中有 A,B 两个账号,现在公司想从账号 A 中取出一万元,存入账号 B。那么就可以定义一个事务,该事务包括两个操作,第一个操作是从账号 A 中减去一万元,第二个操作是向账号 B 中加入一万元。这两个操作要么全做,要么全不做。全做或者全不做,数据库都处于一致性状态。如果只做一个操作则用户逻辑上就会发生错误,少了一万元,这时数据库就处于不一致性状态。可见一致性与原子性是密切相关的。

3. 隔离性

一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对其他并发事务是隔离的,并发执行的各个事务之间不能互相干扰。

4. 持续性

持续性也称永久性(Permanence),指一个事务一旦提交,它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其执行结果有任何影响。

事务是恢复和并发控制的基本单位。所以下面的讨论均以事务为对象。

保证事务 ACID 特性是事务管理的重要任务。事务 ACID 特性可能遭到破坏的因素有:

(1) 多个事务并行运行时,不同事务的操作交叉执行;

(2) 事务在运行过程中被强行停止。

在第一种情况下,数据库管理系统必须保证多个事务的交叉运行不影响这些事务的原子性。在第二种情况下,数据库管理系统必须保证被强行终止的事务对数据库和其他事务没有任何影响。

这些就是数据库管理系统中恢复机制和并发控制机制的责任。

10.2 数据库恢复概述

尽管数据库系统中采取了各种保护措施来防止数据库的安全性和完整性被破坏,保证并发事务的正确执行,但是计算机系统中硬件的故障、软件的错误、操作员的失误以及恶意的破坏仍是不可避免的,这些故障轻则造成运行事务非正常中断,影响数据库中数据的正确性,重则破坏数据库,使数据库中全部或部分数据丢失。因此数据库管理系统必须具有把数

据库从错误状态恢复到某一已知的正确状态(亦称为一致状态或完整状态)的功能,这就是数据库的恢复。恢复子系统是数据库管理系统的一个重要组成部分,而且还相当庞大,常常占整个系统代码的百分之十以上。数据库系统所采用的恢复技术是否行之有效,不仅对系统的可靠程度起着决定性作用,而且对系统的运行效率也有很大影响,是衡量系统性能优劣的重要指标。

10.3 故障的种类

数据库系统中可能发生各种各样的故障,大致可以分以下几类:

一、事务内部的故障

事务内部的故障有的是可以通过事务程序本身发现的(见下面转账事务的例子),有的是非预期的,不能由事务程序处理的。

例如,银行转账事务,这个事务把一笔金额从一个账户甲转给另一个账户乙。

```
BEGIN TRANSACTION
  读账户甲的余额 BALANCE;
  BALANCE = BALANCE-AMOUNT;(AMOUNT 为转账金额)
  IF (BALANCE < 0) THEN
    { 打印"金额不足,不能转账";
      ROLLBACK;(撤销刚才的修改,恢复事务)}
  ELSE
    { 读账户乙的余额 BALANCE1;
      BALANCE1 = BALANCE1 + AMOUNT;
      写回 BALANCE1;
      COMMIT;}
```

这个例子所包括的两个更新操作要么全部完成要么全部不做。否则就会使数据库处于不一致状态,例如只把账户甲的余额减少了而没有把账户乙的余额增加。

在这段程序中若产生账户甲余额不足的情况,应用程序可以发现并让事务滚回,撤销已作的修改,恢复数据库到正确状态。

事务内部更多的故障是非预期的,是不能由应用程序处理的。如运算溢出、并发事务发生死锁而被选中撤销该事务、违反了某些完整性限制等。以后,事务故障仅指这类非预期的故障。

事务故障意味着事务没有达到预期的终点(COMMIT 或者显式的 ROLLBACK),因此,数据库可能处于不正确状态。恢复程序要在不影响其他事务运行的情况下,强行回滚(ROLLBACK)该事务,即撤销该事务已经作出的任何对数据库的修改,使得该事务好像根本没有启动一样。

这类恢复操作称为事务撤销(UNDO)。

二、系统故障

系统故障是指造成系统停止运转的任何事件，使得系统要重新启动。例如，特定类型的硬件错误(CPU 故障)、操作系统故障、DBMS 代码错误、系统断电等。这类故障影响正在运行的所有事务，但不破坏数据库。这时主存内容，尤其是数据库缓冲区(在内存)中的内容都被丢失，所有运行事务都非正常终止。发生系统故障时，一些尚未完成的事务的结果可能已送入物理数据库，从而造成数据库可能处于不正确的状态。为保证数据一致性，需要清除这些事务对数据库的所有修改。

恢复子系统必须在系统重新启动时让所有非正常终止的事务回滚，强行撤销(UNDO)所有未完成事务。

另一方面，发生系统故障时，有些已完成的事务可能有一部分甚至全部留在缓冲区，尚未写回到磁盘上的物理数据库中，系统故障使得这些事务对数据库的修改部分或全部丢失，这也会使数据库处于不一致状态，因此应将这些事务已提交的结果重新写入数据库。所以系统重新启动后，恢复子系统除需要撤销所有未完成的事务外，还需要重做(REDO)所有已提交的事务，以将数据库真正恢复到一致状态。

三、介质故障

系统故障常称为软故障(Soft Crash)，介质故障称为硬故障(Hard Crash)。硬故障指外存故障，如磁盘损坏、磁头碰撞，瞬时强磁场干扰等。这类故障将破坏数据库或部分数据库，并影响正在存取这部分数据的所有事务。这类故障比前两类故障发生的可能性小得多，但破坏性最大。

四、计算机病毒

计算机病毒是一种人为的故障或破坏，是一些恶作剧者研制的一种计算机程序。这种程序与其他程序不同，它像微生物学所称的病毒一样可以繁殖和传播，并造成对计算机系统包括数据库的危害。

病毒的种类很多，不同病毒有不同的特征。小的病毒只有 20 条指令，不到 50 B。大的病毒像一个操作系统，由上万条指令组成。

有的病毒传播很快，一旦侵入系统就马上摧毁系统；有的病毒有较长的潜伏期，机器在感染后数天或数月才开始发病；有的病毒感染系统所有的程序和数据；有的只对某些特定的程序和数据感兴趣。多数病毒一开始并不摧毁整个计算机系统，它们只在数据库中或其他数据文件中将小数点向左或向右移一移，增加或删除一两个“0”。

计算机病毒已成为计算机系统的主要威胁，自然也是数据库系统的主要威胁。为此计算机的安全工作者已研制了许多预防病毒的“疫苗”，检查、诊断、消灭计算机病毒的软件也在不断发展。但是，至今还没有一种可以使计算机“终生”免疫的疫苗。因此数据库一旦被破坏仍要用恢复技术把数据库加以恢复。

总结各类故障，对数据库的影响有两种可能性。一是数据库本身被破坏。二是数据库

没有被破坏，但数据可能不正确，这是由于事务的运行被非正常终止造成的。

恢复的基本原理十分简单。可以用一个词来概括：**冗余**。这就是说，数据库中任何一部分被破坏的或不正确的数据可以根据存储在系统别处的冗余数据来重建。尽管恢复的基本原理很简单但实现技术的细节却相当复杂，下面略去一些细节，介绍数据库恢复的实现技术。

10.4 恢复的实现技术

恢复机制涉及的两个关键问题是：第一，如何建立冗余数据；第二，如何利用这些冗余数据实施数据库恢复。

建立冗余数据最常用的技术是数据转储和登录日志文件。通常在一个数据库系统中，这两种方法是一起使用的。

10.4.1 数据转储

数据转储是数据库恢复中采用的基本技术。所谓转储即 DBA 定期地将整个数据库复制到磁带或另一个磁盘上保存起来的过程。这些备用的数据称为后备副本或后援副本。

当数据库遭到破坏后可以将后备副本重新装入，但重装后备副本只能将数据库恢复到转储时的状态，要想恢复到故障发生时的状态，必须重新运行自转储以后的所有更新事务。例如，在图 10.1 中，系统在 T_a 时刻停止运行事务，进行数据库转储，在 T_b 时刻转储完毕，得到 T_b 时刻的数据库一致性副本。系统运行到 T_f 时刻发生故障。为恢复数据库，首先由 DBA 重装数据库后备副本，将数据库恢复至 T_b 时刻的状态，然后重新运行自 $T_b \sim T_f$ 时刻的所有更新事务，这样就把数据库恢复到故障发生前的一致状态。

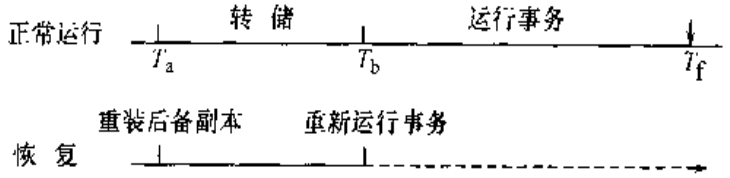


图 10.1 转储和恢复

转储是十分耗费时间和资源的，不能频繁进行。DBA 应该根据数据库使用情况确定一个适当的转储周期。

转储可分为静态转储和动态转储。

静态转储是在系统中无运行事务时进行的转储操作。即转储操作开始的时刻，数据库处于一致性状态，而转储期间不允许（或不存在）对数据库的任何存取、修改活动。显然，静态转储得到的一定是一个数据一致性的副本。

静态转储简单，但转储必须等待正运行的用户事务结束才能进行。同样，新的事务必须

等待转储结束才能执行。显然,这会降低数据库的可用性。

动态转储是指转储期间允许对数据库进行存取或修改。即转储和用户事务可以并发执行。

动态转储可以克服静态转储的缺点,它不用等待正在运行的用户事务结束,也不会影响新事务的运行。但是,转储结束时后援副本上的数据并不能保证正确有效。例如,在转储期间的某个时刻 T_c,系统把数据 A = 100 转储到磁带上,而在下一时刻 T_d,某一事务将 A 改为 200。转储结束后,后备副本上的 A 已是过时的数据了。

为此,必须把转储期间各事务对数据库的修改活动登记下来,建立日志文件(log file)。这样,后援副本加上日志文件就能把数据库恢复到某一时刻的正确状态。

转储还可以分为海量转储和增量转储两种方式。海量转储是指每次转储全部数据库。增量转储则指每次只转储上一次转储后更新过的数据。从恢复角度看,使用海量转储得到的后备副本进行恢复一般说来会更方便些。但如果数据库很大,事务处理又十分频繁,则增量转储方式更实用更有效。

数据转储有两种方式,分别可以在两种状态下进行,因此数据转储方法可以分为四类:动态海量转储、动态增量转储、静态海量转储和静态增量转储,如表 10.1 所示。

表 10.1 数据转储分类

		转储状态	
		动态转储	静态转储
转储方式	海量转储	动态海量转储	静态海量转储
	增量转储	动态增量转储	静态增量转储

10.4.2 登记日志文件(Logging)

一、日志文件的格式和内容

日志文件是用来记录事务对数据库的更新操作的文件。不同数据库系统采用的日志文件格式并不完全一样。概括起来日志文件主要有两种格式:以记录为单位的日志文件和以数据块为单位的日志文件。

对于以记录为单位的日志文件,日志文件中需要登记的内容包括:

- 各个事务的开始(BEGIN TRANSACTION)标记;
- 各个事务的结束(COMMIT 或 ROLLBACK)标记;
- 各个事务的所有更新操作。

这里每个事务开始的标记、每个事务的结束标记和每个更新操作均作为日志文件中的一个日志记录(log record)。

每个日志记录的内容主要包括:

- 事务标识(标明是哪个事务);
- 操作的类型(插入、删除或修改);
- 操作对象(记录内部标识);
- 更新前数据的旧值(对插入操作而言,此项为空值);
- 更新后数据的新值(对删除操作而言,此项为空值)。

对于以数据块为单位的日志文件,日志记录的内容包括事务标识和被更新的数据块。由于将更新前的整个块和更新后的整个块都放入日志文件中,操作的类型和操作对象等信息就不必放入日志记录中。

二、日志文件的作用

日志文件在数据库恢复中起着非常重要的作用。可以用来进行事务故障恢复和系统故障恢复,并协助后备副本进行介质故障恢复。具体作用是:

(1) 事务故障恢复和系统故障恢复必须用日志文件。

(2) 在动态转储方式中必须建立日志文件,后备副本和日志文件结合起来才能有效地恢复数据库。

(3) 在静态转储方式中,也可以建立日志文件。当数据库毁坏后可重新装入后援副本把数据库恢复到转储结束时刻的正确状态,然后利用日志文件,把已完成的事务进行重做处理,对故障发生时尚未完成的事务进行撤销处理。这样不必重新运行那些已完成的事务程序就可把数据库恢复到故障前某一时刻的正确状态,如图 10.2 所示。

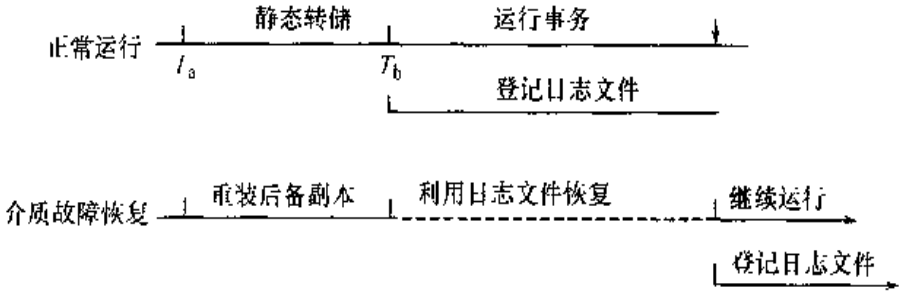


图 10.2 利用日志文件恢复

三、登记日志文件

为保证数据库是可恢复的,登记日志文件时必须遵循两条原则:

(1) 登记的次序严格按并发事务执行的时间次序;

(2) 必须先写日志文件,后写数据库。

把对数据的修改写到数据库中和把表示这个修改的日志记录写到日志文件中是两个不同的操作。有可能在这两个操作之间发生故障,即这两个写操作只完成了一个。如果先写了数据库修改,而在运行记录中没有登记这个修改,则以后就无法恢复这个修改了。如果先写日志,但没有修改数据库,按日志文件恢复时只不过是多执行一次不必要的 UNDO 操作,并不会影响数据库的正确性。所以为了安全,一定要先写日志文件,即首先把日志记录写到

日志文件中，然后写数据库的修改。这就是“先写日志文件”的原则。

10.5 恢复策略

当系统运行过程中发生故障，利用数据库后备副本和日志文件就可以将数据库恢复到故障前的某个一致性状态。不同故障其恢复策略和方法也不一样。

10.5.1 事务故障的恢复

事务故障是指事务在运行至正常终止点前被终止，这时恢复子系统应利用日志文件撤销(UNDO)此事务已对数据库进行的修改。事务故障的恢复是由系统自动完成的，对用户是透明的。系统的恢复步骤是：

(1) 反向扫描日志文件(即从最后向前扫描日志文件)，查找该事务的更新操作。

(2) 对该事务的更新操作执行逆操作。即将日志记录中“更新前的值”写入数据库。这样，如果记录中是插入操作，则相当于做删除操作(因此时“更新前的值”为空)；若记录中是删除操作，则做插入操作；若是修改操作，则相当于用修改前值代替修改后值。

(3) 继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理。

(4) 如此处理下去，直至读到此事务的开始标记，事务故障恢复就完成了。

10.5.2 系统故障的恢复

前面已讲过，系统故障造成数据库不一致状态的原因有两个，一是未完成事务对数据库的更新可能已写入数据库，二是已提交事务对数据库的更新可能还留在缓冲区没来得及写入数据库。因此恢复操作就是要撤销故障发生时未完成的事务，重做已完成的事务。

系统故障的恢复是由系统在重新启动时自动完成的，不需要用户干预。

系统的恢复步骤是：

(1) 正向扫描日志文件(即从头扫描日志文件)，找出在故障发生前已经提交的事务(这些事务既有 BEGIN TRANSACTION 记录，也有 COMMIT 记录)，将其事务标识记入重做(REDO)队列。同时找出故障发生时尚未完成的事务(这些事务只有 BEGIN TRANSACTION 记录，无相应的 COMMIT 记录)，将其事务标识记入撤销队列。

(2) 对撤销队列中的各个事务进行撤销(UNDO)处理。

进行 UNDO 处理的方法是，反向扫描日志文件，对每个 UNDO 事务的更新操作执行逆操作，即将日志记录中“更新前的值”写入数据库。

(3) 对重做队列中的各个事务进行重做(REDO)处理。

进行 REDO 处理的方法是：正向扫描日志文件，对每个 REDO 事务重新执行日志文件登记的操作。即将日志记录中“更新后的值”写入数据库。

10.5.3 介质故障的恢复

发生介质故障后，磁盘上的物理数据和日志文件被破坏，这是最严重的一种故障，恢复方法是重装数据库，然后重做已完成的事务。

1. 装入最新的数据库后备副本（离故障发生时刻最近的转储副本），使数据库恢复到最近一次转储时的一致性状态。

对于动态转储的数据库副本，还需同时装入转储开始时刻的日志文件副本，利用恢复系统故障的方法（即 REDO + UNDO），才能将数据库恢复到一致性状态。

2. 装入相应的日志文件副本（转储结束时刻的日志文件副本），重做已完成的事务。即首先扫描日志文件，找出故障发生时已提交的事务的标识，将其记入重做队列。然后正向扫描日志文件，对重做队列中的所有事务进行重做处理。即将日志记录中“更新后的值”写入数据库。

这样就可以将数据库恢复至故障前某一时刻的一致状态了。

介质故障的恢复需要 DBA 介入。但 DBA 只需要重装最近转储的数据库副本和有关的各日志文件副本，然后执行系统提供的恢复命令即可，具体的恢复操作仍由 DBMS 完成。

10.6 具有检查点的恢复技术

利用日志技术进行数据库恢复时，恢复子系统必须搜索日志，确定哪些事务需要 REDO，哪些事务需要 UNDO。一般来说，需要检查所有日志记录。这样做具有两个问题。一是搜索整个日志将耗费大量的时间。二是很多需要 REDO 处理的事务实际上已经将它们的更新操作结果写到数据库中了，然而恢复子系统又重新执行了这些操作，浪费了大量时间。为了解决这些问题，又发展了具有检查点的恢复技术。这种技术在日志文件中增加一类新的记录——检查点（checkpoint）记录，增加一个重新开始文件，并让恢复子系统在登录日志文件期间动态地维护日志。

检查点记录的内容包括：

① 建立检查点时刻所有正在执行的事务清单；

② 这些事务最近一个日志记录的地址。

重新开始文件用来记录各个检查点记录在日志文件中的地址。图 10.3 说明了建立检查点 C_i 时对应的日志文件和重新开始文件。

动态维护日志文件的方法是，周期性地执行如下操作：建立检查点，保存数据库状态。具体步骤是：

① 将当前日志缓冲区中的所有日志记录写入磁盘的日志文件上；

② 在日志文件中写入一个检查点记录；

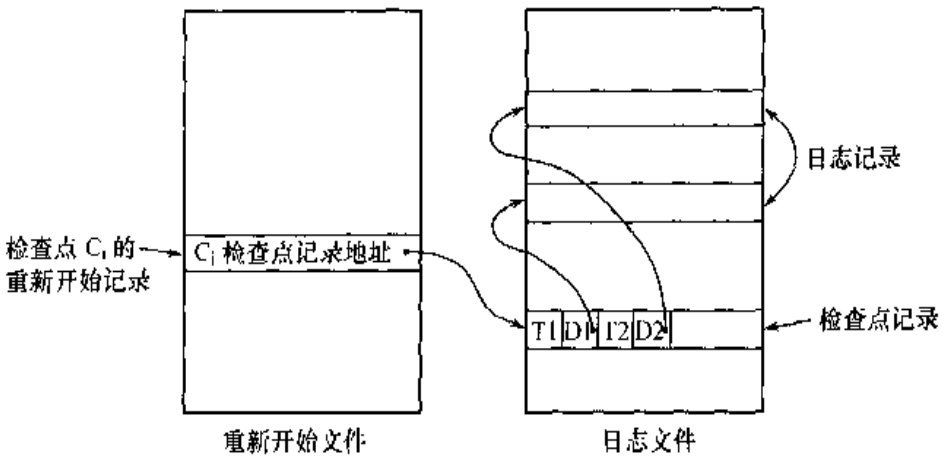


图 10.3 具有检查点的日志文件和重新开始文件

③ 将当前数据缓冲区的所有数据记录写入磁盘的数据库中；

④ 把检查点记录在日志文件中的地址写入一个重新开始文件。

恢复子系统可以定期或不定期地建立检查点保存数据库状态。检查点可以按照预定的一个时间间隔建立，如每隔一小时建立一个检查点；也可以按照某种规则建立检查点，如日志文件已写满一半建立一个检查点。

使用检查点方法可以改善恢复效率。当事务 T 在一个检查点之前提交，T 对数据库所做的修改一定都已写入数据库，写入时间是在这个检查点建立之前或在这个检查点建立之时。这样，在进行恢复处理时，没有必要对事务 T 执行 REDO 操作。

系统出现故障时，恢复子系统将根据事务的不同状态采取不同的恢复策略，如图 10.4 所示。

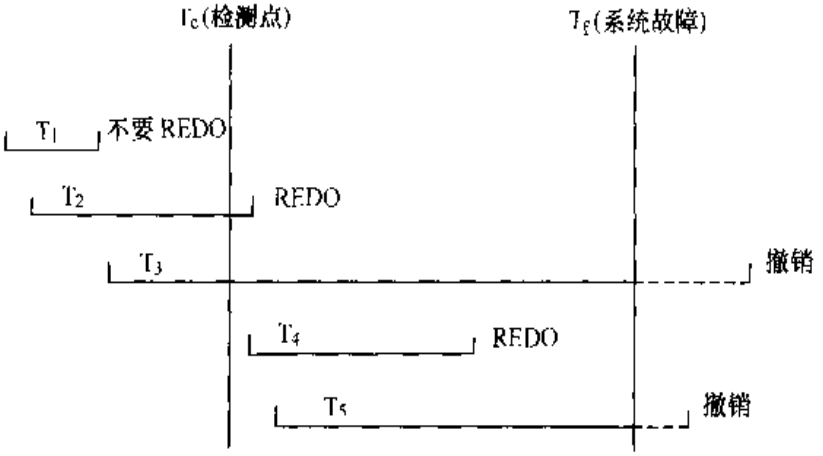


图 10.4 恢复子系统采取的不同策略

T_1：在检查点之前提交；

T_2：在检查点之前开始执行，在检查点之后故障点之前提交；

T_3:在检查点之前开始执行,在故障点时还未完成;

T_4:在检查点之后开始执行,在故障点之前提交;

T_5:在检查点之后开始执行,在故障点时还未完成。

T_3 和 T_5 在故障发生时还未完成,所以予以撤销;T_2 和 T_4 在检查点之后才提交,它们对数据库所做的修改在故障发生时可能还在缓冲区中,尚未写入数据库,所以要 REDO;T_1 在检查点之前已提交,所以不必执行 REDO 操作。

系统使用检查点方法进行恢复的步骤是:

① 从重新开始文件中找到最后一个检查点记录在日志文件中的地址,由该地址在日志文件中找到最后一个检查点记录。

② 由该检查点记录得到检查点建立时刻所有正在执行的事务清单 ACTIVE-LIST。

这里建立两个事务队列:

- UNDO-LIST:需要执行 undo 操作的事务集合;
- REDO-LIST:需要执行 redo 操作的事务集合。

把 ACTIVE-LIST 暂时放入 UNDO-LIST 队列,REDO 队列暂为空。

③ 从检查点开始正向扫描日志文件。

- 如有新开始的事务 T_i,把 T_i 暂时放入 UNDO-LIST 队列;
- 如有提交的事务 T_j,把 T_j 从 UNDO-LIST 队列移到 REDO-LIST 队列;直到日志文件结束。

④ 对 UNDO-LIST 中的每个事务执行 UNDO 操作,对 REDO-LIST 中的每个事务执行 REDO 操作。

10.7 数据库镜像

根据前面所述,介质故障是对系统影响最为严重的一种故障。系统出现介质故障后,用户应用全部中断,恢复起来也比较费时。而且 DBA 必须周期性地转储数据库,这也加重了 DBA 的负担。如果不及时而正确地转储数据库,一旦发生介质故障,会造成较大的损失。

随着磁盘容量越来越大,价格越来越便宜,为避免磁盘介质出现故障影响数据库的可用性,许多数据库管理系统提供了数据库镜像(Mirror)功能用于数据库恢复。即根据 DBA 的要求,自动把整个数据库或其中的关键数据复制到另一个磁盘上。每当主数据库更新时,DBMS 自动把更新后的数据复制过去,即 DBMS 自动保证镜像数据与主数据库的一致性(如图 10.5(a)所示)。这样,一旦出现介质故障,可由镜像磁盘继续提供使用,同时 DBMS 自动利用镜像磁盘数据进行数据库的恢复,不需要关闭系统和重装数据库副本(如图 10.5(b)所示)。在没有出现故障时,数据库镜像还可以用于并发操作,即当一个用户对数据加排它锁修改数据时,其他用户可以读镜像数据库上的数据,而不必等待该用户释放锁。

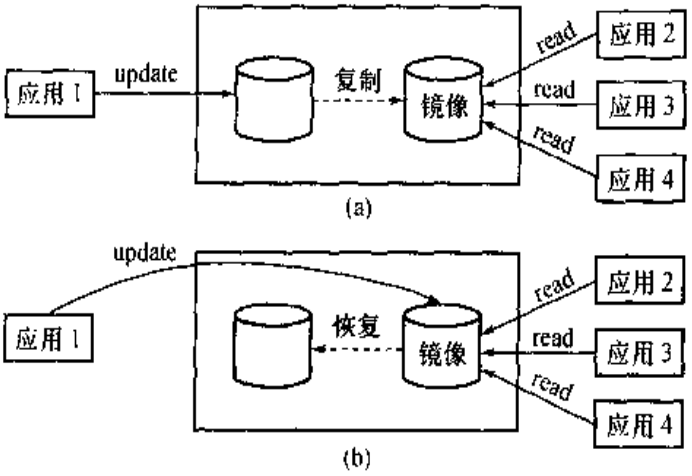


图 10.5　数据库镜像

由于数据库镜像是通过复制数据实现的，频繁地复制数据自然会降低系统运行效率，因此在实际应用中用户往往只选择对关键数据和日志文件镜像，而不是对整个数据库进行镜像。

10.8　小　　结

保证数据一致性是对数据库的最基本的要求。事务是数据库的逻辑工作单位，只要DBMS能够保证系统中一切事务的原子性、一致性、隔离性和持续性，也就保证了数据库处于一致状态。为了保证事务的原子性、一致性与持续性，DBMS必须对事务故障、系统故障和介质故障进行恢复。数据库转储和登记日志文件是恢复中最经常使用的技术。恢复的基本原理就是利用存储在后备副本、日志文件和数据库镜像中的冗余数据来重建数据库。

事务不仅是恢复的基本单位，也是并发控制的基本单位，为了保证事务的隔离性和一致性，DBMS需要对并发操作进行控制。

习　　题

1. 试述事务的概念及事务的四个特性。恢复技术能保证事务的哪些特性？
2. 为什么事务非正常结束时会影响数据库数据的正确性？请举例说明之。
3. 数据库中为什么要有恢复子系统？它的功能是什么？
4. 数据库运行中可能产生的故障有哪几类？哪些故障影响事务的正常执行？哪些故障破坏数据库数据？
5. 数据库恢复的基本技术有哪些？
6. 数据库转储的意义是什么？试比较各种数据转储方法。
7. 什么是日志文件？为什么要设立日志文件？

8. 登记日志文件时为什么必须先写日志文件,后写数据库?

9. 针对不同的故障,试给出恢复的策略和方法 (即如何进行事务故障的恢复? 系统故障的恢复? 介质故障恢复?)

10. 什么是检查点记录? 检查点记录包括哪些内容?

11. 具有检查点的恢复技术有什么优点? 试举一个具体的例子加以说明。

12. 试述使用检查点方法进行恢复的步骤。

13. 什么是数据库镜像? 它有什么用途?

*14. 试述你了解的某一个实际的 DBMS 产品中采用的恢复策略。

*15. 试用恢复的基本技术设计一个恢复子系统,给出这个子系统的恢复策略,包括:

(a) 当产生某一类故障时如何恢复数据库的方法;

(b) 日志文件的结构;

(c) 登记日志文件的方法;

(d) 利用日志文件恢复事务的方法;

(e) 转储的类型;

(f) 转储的后备副本和日志文件如何配合使用。

本章参考文献

1 Davies, Jr C T. Recovery Semantics for a DB/DC System. Proceedings of the ACM Annual Conference, 1973

2 Bjork L A. Recovery Scenario for a DB/DC System. Proceedings of the ACM Annual Conference, 1973

(文献[1]和[2]是系统恢复领域中两篇最早的论文。)

3 Theo Haerder and Andreas Reuter. Principles of Transaction-Oriented Database Recovey. ACM Computing Surveys, Vol. 15, No. 4, 1983

4 Gray Jim, et al. The Recovery Manager of the System R Data Manager. ACM Computing Surveys, Vol. 13, No. 2, 1981 (本文概述了 IBM 的数据库管理系统(实验系统)System R 的恢复技术。)

5 Crus R A. Data Recovery in IBM Database 2. IBM Sys, J. 23, No. 2, 1984

(本文介绍了 IBM DB2 的恢复技术。)

6 Lorie R A. Physical Integrity in a Large Segmented Database. ACM Transactions on Database Systems, Volume 2, Number 1, 1977 (本文研究了 System R 的影子页面技术。)

7 Chandy K M, Browne J C, Dissley C W, et al. Analytic Models for Rollback and Recovery Strategies in Database Systems. IEEE Transactions on Software Engineering, Vol. SE-1, Number 1, 1975

(本文提出了数据库系统恢复和回滚的分析模型。)

8 Reuter A. A Fast Transaction-Oriented Logging Scheme for UNDO Recovery. IEEE Transactions on Software Engineering, Volume SE-6, Number 4, 1980

(本文给出了一个快速处理 UNDO 操作的日志模式。)

9 Lilien L, Bhargava B. Database Integrity Block Construct: Concept and Design Issues. IEEE Transactions on Software Engineering, Vol. SE-11, No. 9, 1985

(本文介绍了完整性、并发控制和系统恢复相结合的问题。)

10 Verhofstad J S M. Recovery Techniques for Database Systems. ACM Computing Surveys, Volume 10, Number 2, 1978

（本文系统地综述了数据库管理系统的恢复技术。）

11 Bernstein A, Hadzilacos V, Goodman W. Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987

（本书包含了有关 DBMS 恢复的全面介绍。）

12 Mohan C, Haderle D, Lindsay R et al. ARIES: A Transaction Recovery Method Supporting Fine Granularity Locking and Partial Rollback Using Write-Ahead Logging. ACM Transactions on Database Systems, 1991.

（本文讨论了 ARIES 系统中事务恢复技术。）

13 Mohan C. Commit-LSN: A Novel and Simple Method for Reducing Locking and Latching in Transaction Processing Systems. Proceedings of the International Conference on Very Large Data Bases, 1990

14 Korth H F, Speegle G. Long Duration Transactions in Software Design Projects. Proceedings of the International Conference on Data Engineering, 1990

（本文讨论了长事务的问题。）

15 Korth H F, Levy E. Silberschatz A. A Formal Approach to Recovery by Compensating Transactions. Proceedings of the International Conference on Very Large Data Bases, 1990

（本文讨论了长事务的恢复问题。）

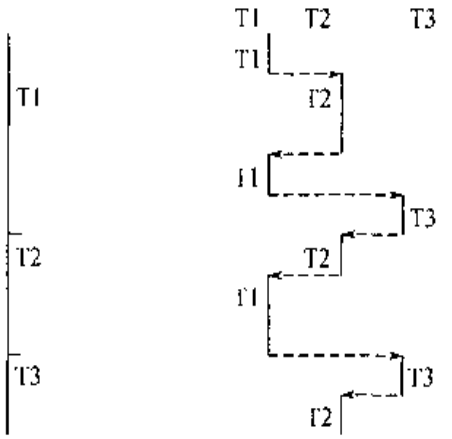
第十一章 并发控制

数据库是一个共享资源，可以供多个用户使用。允许多个用户同时使用的数据库系统称为多用户数据库系统。例如飞机定票数据库系统、银行数据库系统等都是多用户数据库系统。在这样的系统中，在同一时刻并发运行的事务数可达数百个。

事务可以一个一个地串行执行，即每个时刻只有一个事务运行，其他事务必须等到这个事务结束以后方能运行，如图 11.1(a)所示。事务在执行过程中需要不同的资源，有时需要CPU，有时需要存取数据库，有时需要 I/O，有时需要通信。如果事务串行执行，则许多系统资源将处于空闲状态。因此，为了充分利用系统资源发挥数据库共享资源的特点，应该允许多个事务并行地执行。

在单处理机系统中，事务的并行执行实际上是这些并行事务的并行操作轮流交叉运行，如图 11.1(b)所示。这种并行执行方式称为**交叉并发方式**(Interleaved Concurrency)。虽然单处理机系统中的并行事务并没有真正地并行运行，但是减少了处理机的空闲时间，提高了系统的效率。

在多处理机系统中，每个处理机可以运行一个事务，多个处理机可以同时运行多个事务，实现多个事务真正的并行运行。这种并行执行方式称为**同时并发方式**(Simultaneous Concurrency)。本章讨论的数据库系统并发控制技术是以单处理机系统为基础的。这些理论可以推广到多处理机的情况。



(a) 事务的串行执行方式　(b) 事务的交叉并发执行方式

图 11.1　事务的执行方式

当多个用户并发地存取数据库时就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会存取和存储不正确的数据，破坏事务的一致性和数据库的一致性。所以数据库管理系统必须提供并发控制机制。并发控制机制是衡量一个数据库管理系统性

能的重要标志之一。

11.1 并发控制概述

在第十章中已经讲到,事务是并发控制的基本单位,保证事务 ACID 特性是事务处理的重要任务,而事务 ACID 特性可能遭到破坏的原因之一是多个事务对数据库的并发操作造成的。为了**保证事务的隔离性和一致性**,DBMS 需要对并发操作进行正确调度。这些就是数据库管理系统中并发控制机制的责任。

下面先来看一个例子,说明并发操作带来的数据的不一致性问题。

[例 1] 考虑飞机订票系统中的一个活动序列:

① 甲售票点(甲事务)读出某航班的机票余额 A,设 A = 16;

② 乙售票点(乙事务)读出同一航班的机票余额 A,也为 16;

③ 甲售票点卖出一张机票,修改余额 A←A - 1,所以 A 为 15,把 A 写回数据库;

④ 乙售票点也卖出一张机票,修改余额 A← A - 1,所以 A 为 15,把 A 写回数据库。

结果明明卖出两张机票,数据库中机票余额只减少 1。

这种情况称为数据库的不一致性。这种不一致性是由并发操作引起的。在并发操作情况下,对甲、乙两个事务的操作序列的调度是随机的。若按上面的调度序列执行,甲事务的修改就被丢失。这是由于第 4 步中乙事务修改 A 并写回后覆盖了甲事务的修改。

下面把事务读数据 x 记为 R(x),写数据 x 记为 W(x)。

仔细分析并发操作带来的数据不一致性主要包括丢失修改、不可重复读和读"脏"数据等,如图 11.2 所示。

T_1	T_2
① R(A) = 16	
②	R(A) = 16
③ A←A - 1 W(A) = 15	
④	A←A - 1 W(A) = 15

(a) 丢失修改

T_1	T_2
① R(A) = 50 R(B) = 100 求和 = 150	
②	R(B) = 100 B←B * 2 W(B) = 200
③ R(A) = 50 R(B) = 200 和 = 250 (验算不对)	

(b) 不可重复读

T_1	T_2
① R(C) = 100 C←C * 2 W(C) = 200	
②	R(C) = 200
③ ROLLBACK C 恢复为 100	

(c) 读"脏"数据

图 11.2 三种数据不一致性

1. 丢失修改(Lost Update)

两个事务 T_1 和 T_2 读入同一数据并修改，T_2 提交的结果破坏了 T_1 提交的结果，导致 T_1 的修改被丢失，如图 11.2(a)所示。上面飞机订票例子就属此类。

2. 不可重复读(Non-Repeatable Read)

不可重复读是指事务 T_1 读取数据后，事务 T_2 执行更新操作，使 T_1 无法再现前一次读取结果。具体地讲，不可重复读包括三种情况：

(1) 事务 T_1 读取某一数据后，事务 T_2 对其做了修改，当事务 T_1 再次读该数据时，得到与前一次不同的值。例如在图 11.2(b)中，T_1 读取 B = 100 进行运算，T_2 读取同一数据 B，对其进行修改后将 B = 200 写回数据库。T_1 为了对读取值校对重读 B，B 已为 200，与第一次读取值不一致。

(2) 事务 T_1 按一定条件从数据库中读取了某些数据记录后，事务 T_2 删除了其中部分记录，当 T_1 再次按相同条件读取数据时，发现某些记录神秘地消失了。

(3) 事务 T_1 按一定条件从数据库中读取某些数据记录后，事务 T_2 插入了一些记录，当 T_1 再次按相同条件读取数据时，发现多了一些记录。

后两种不可重复读有时也称为幻影(Phantom Row)现象。

3. 读"脏"数据(Dirty Read)

读"脏"数据是指事务 T_1 修改某一数据，并将其写回磁盘，事务 T_2 读取同一数据后，T_1 由于某种原因被撤销，这时 T_1 已修改过的数据恢复原值，T_2 读到的数据就与数据库中的数据不一致，则 T_2 读到的数据就为"脏"数据，即不正确的数据。例如在图 11.2(c)中 T_1 将 C 值修改为 200，T_2 读到 C 为 200，而 T_1 由于某种原因撤销，其修改作废，C 恢复原值 100，这时 T_2 读到的 C 为 200，与数据库内容不一致，就是"脏"数据。

产生上述三类数据不一致性的主要原因是并发操作破坏了事务的隔离性。并发控制就是要用正确的方式调度并发操作，使一个用户事务的执行不受其他事务的干扰，从而避免造成数据的不一致性。

另一方面，对数据库的应用有时允许某些不一致性，例如有些统计工作涉及数据量很大，读到一些"脏"数据对统计精度没什么影响，这时可以降低对一致性的要求以减少系统开销。

并发控制的主要技术有封锁(Locking)、时间戳(Timestamp)和乐观控制法，商用的 DBMS 一般都采用封锁方法。例如，在[例 1]中，甲事务要修改 A，若在读出 A 前先锁住 A，其他事务就不能再读取和修改 A 了，直到甲修改并写回 A 后解除了对 A 的封锁为止。这样，就不会丢失甲的修改。

11.2 封 锁

封锁是实现并发控制的一个非常重要的技术。所谓封锁就是事务 T 在对某个数据对象

例如表、记录等操作之前，先向系统发出请求，对其加锁。加锁后事务 T 就对该数据对象有了一定的控制，在事务 T 释放它的锁之前，其他的事务不能更新此数据对象。

确切的控制由封锁的类型决定。基本的封锁类型有两种：排它锁(Exclusive Locks，简称 X 锁) 和共享锁(Share Locks，简称 S 锁)。

排它锁又称为**写锁**。若事务 T 对数据对象 A 加上 X 锁，则只允许 T 读取和修改 A，其他任何事务都不能再对 A 加任何类型的锁，直到 T 释放 A 上的锁。这就保证了其他事务在 T 释放 A 上的锁之前不能再读取和修改 A。

共享锁又称为**读锁**。若事务 T 对数据对象 A 加上 S 锁，则事务 T 可以读 A 但不能修改 A，其他事务只能再对 A 加 S 锁，而不能加 X 锁，直到 T 释放 A 上的 S 锁。这就保证了其他事务可以读 A，但在 T 释放 A 上的 S 锁之前不能对 A 做任何修改。

排它锁与共享锁的控制方式可以用图 11.3 的相容矩阵来表示。

T_1 \ T_2	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

Y = Yes，相容的请求
N = No，不相容的请求

图 11.3 封锁类型的相容矩阵

在图 11.3 的封锁类型相容矩阵中，最左边一列表示事务 T_1 已经获得的数据对象上的锁的类型，其中横线表示没有加锁。最上面一行表示另一事务 T_2 对同一数据对象发出的封锁请求。T_2 的封锁请求能否被满足用矩阵中的 Y 和 N 表示，其中 Y 表示事务 T_2 的封锁要求与 T_1 已持有的锁相容，封锁请求可以满足。N 表示 T_2 的封锁请求与 T_1 已持有的锁冲突，T_2 的请求被拒绝。

[例 2] 图 11.4 给出了使用封锁机制解决丢失修改、不可重复读和读“脏”数据等不一致性问题的例子。

图 11.4(a)中，事务 T_1 在读 A 进行修改之前先对 A 加 X 锁，当 T_2 再请求对 A 加 X 锁时被拒绝，T_2 只能等待 T_1 释放 A 上的锁后 T_2 获得对 A 的 X 锁，这时它读到的 A 已经是 T_1 更新过的值 15，再按此新的 A 值进行运算，并将结果值 A = 14 送回到磁盘。这样就避免了丢失 T_1 的更新。

图 11.4(b)中，事务 T_1 在读 A，B 之前，先对 A，B 加 S 锁，这样其他事务只能再对 A，B 加 S 锁，而不能加 X 锁，即其他事务只能读 A，B，而不能修改它们。所以当 T_2 为修改 B 而申请对 B 的 X 锁时被拒绝只能等待 T_1 释放 B 上的锁。T_1 为验算再读 A，B，这时读出的 B 仍是 100，求和结果仍为 150，即可重复读。T_1 结束才释放 A，B 上的 S 锁。T_2 才获得对 B 的 X 锁，

图 11.4(c)中,事务 T_1 在对 C 进行修改之前,先对 C 加 X 锁,修改其值后写回磁盘。这时 T_2 请求在 C 上加 S 锁,因 T_1 已在 C 上加了 X 锁,T_2 只能等待。T_1 因某种原因被撤销,C 恢复为原值 100,T_1 释放 C 上的 X 锁后 T_2 获得 C 上的 S 锁,读 C = 100。这就避免了 T_2 读"脏"数据。

T_1	T_2
① Xlock A	
② R(A) = 16	
	Xlock A
③ A← A - 1	等待
W(A) = 15	等待
Commit	等待
Unlock A	等待
④	获得 Xlock A
	R(A) = 15
	A←A - 1
⑤	W(A) = 14
	Commit
	Unlock A

(a) 没有丢失修改

T_1	T_2
① Slock A	
Slock B	
R(A) = 50	
R(B) = 100	
求和 = 150	
②	
	Xlock B
	等待
	等待
③ R(A) = 50	等待
R(B) = 100	等待
求和 = 150	等待
Commit	等待
Unlock A	等待
Unlock B	等待
④	获得 XlockB
	R(B) = 100
	B←B * 2
⑤	W(B) = 200
	Commit
	Unlock B

(b) 可重复读

T_1	T_2
Xlock C	
R(C) = 100	
C←C * 2	
W(C) = 200	
②	Slock C
	等待
② ROLLBACK	等待
(C 恢复为 100)	等待
Unlock C	等待
④	获得 Slock C
	R(C) = 100
⑤	Commit C
	Unlock C

(c) 不读"脏"数据

图 11.4 使用封锁机制解决三种数据不一致性的示例

11.3 活锁和死锁

和操作系统一样,封锁的方法可能引起活锁和死锁等问题。

11.3.1 活锁

如果事务 T_1 封锁了数据 R,事务 T_2 又请求封锁 R,于是 T_2 等待。T_3 也请求封锁 R,当 T_1 释放了 R 上的封锁之后系统首先批准了 T_3 的请求,T_2 仍然等待。然后 T_4 又请求封锁 R,

当 T_3 释放了 R 上的封锁之后系统又批准了 T_4 的请求……T_2 有可能永远等待，这就是活锁的情形，如图 11.5(a)所示。

T_1	T_2	T_3	T_4
lock R	.	.	.
.	lock R	.	.
.	等待	Lock R	.
Unlock	等待	.	Lock R
.	等待	Lock R	等待
.	等待	.	等待
.	等待	Unlock	等待
.	等待	.	Lock R
.	等待	.	.

(a) 活锁

T_1	T_2
lock R_1	.
.	Lock R_2
.	.
Lock R_2	.
等待	.
等待	Lock R_1
等待	等待
等待	等待
.	.

(b) 死锁

图 11.5 死锁与活锁示例

避免活锁的简单方法是采用先来先服务的策略。当多个事务请求封锁同一数据对象时，封锁子系统按请求封锁的先后次序对事务排队，数据对象上的锁一旦释放就批准申请队列中第一个事务获得锁。

11.3.2 死锁

如果事务 T_1 封锁了数据 R_1，T_2 封锁了数据 R_2，然后 T_1 又请求封锁 R_2，因 T_2 已封锁了 R_2，于是 T_1 等待 T_2 释放 R_2 上的锁。接着 T_2 又申请封锁 R_1，因 T_1 已封锁了 R_1，T_2 也只能等待 T_1 释放 R_1 上的锁。这样就出现了 T_1 在等待 T_2，而 T_2 又在等待 T_1 的局面，T_1 和 T_2 两个事务永远不能结束，形成死锁。如图 11.5(b)所示。

死锁的问题在操作系统和一般并行处理中已做了深入研究，目前在数据库中解决死锁问题主要有两类方法，一类方法是采取一定措施来预防死锁的发生，另一类方法是允许发生死锁，采用一定手段定期诊断系统中有无死锁，若有则解除之。

1. 死锁的预防

在数据库中，产生死锁的原因是两个或多个事务都已封锁了一些数据对象，然后又都请求对已为其他事务封锁的数据对象加锁，从而出现死等待。防止死锁的发生其实就是要破坏产生死锁的条件。预防死锁通常有两种方法：

(1) 一次封锁法

一次封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行。图 11.5(b)的例子中，如果事务 T_1 将数据对象 R_1 和 R_2 一次加锁，T_1 就可以执行下去，

而 T_2 等待。T_1 执行完后释放 R_1,R_2 上的锁,T_2 继续执行。这样就不会发生死锁。

一次封锁法虽然可以有效地防止死锁的发生,但也存在问题。第一,一次就将以后要用到的全部数据加锁,势必扩大了封锁的范围,从而降低了系统的并发度。第二,数据库中数据是不断变化的,原来不要求封锁的数据,在执行过程中可能会变成封锁对象,所以很难事先精确地确定每个事务所要封锁的数据对象,为此只能扩大封锁范围,将事务在执行过程中可能要封锁的数据对象全部加锁,这就进一步降低了并发度。

(2) 顺序封锁法

顺序封锁法是预先对数据对象规定一个封锁顺序,所有事务都按这个顺序实行封锁。例如在 B 树结构的索引中,可规定封锁的顺序必须是从根结点开始,然后是下一级的子女结点,逐级封锁。

顺序封锁法可以有效地防止死锁,但也同样存在问题。第一,数据库系统中封锁的数据对象极多,并且随数据的插入、删除等操作而不断地变化,要维护这样的资源的封锁顺序非常困难,成本很高。第二,事务的封锁请求可以随着事务的执行而动态地决定,很难事先确定每一个事务要封锁哪些对象,因此也就很难按规定的顺序去施加封锁。

可见,在操作系统中广为采用的预防死锁的策略并不很适合数据库的特点,因此 DBMS 在解决死锁的问题上普遍采用的是诊断并解除死锁的方法。

2. 死锁的诊断与解除

数据库系统中诊断死锁的方法与操作系统类似,一般使用超时法或事务等待图法。

(1) 超时法

如果一个事务的等待时间超过了规定的时限,就认为发生了死锁。超时法实现简单,但其不足也很明显。一是有可能误判死锁,事务因为其他原因使等待时间超过时限,系统会误认为发生了死锁。二是时限若设置得太长,死锁发生后不能及时发现。

(2) 等待图法

事务等待图是一个有向图 $G=(T,U)$。T 为结点的集合,每个结点表示正运行的事务;U 为边的集合,每条边表示事务等待的情况。若 T_1 等待 T_2,则 T_1,T_2 之间划一条有向边,从 T_1 指向 T_2。如图 11.6 所示。

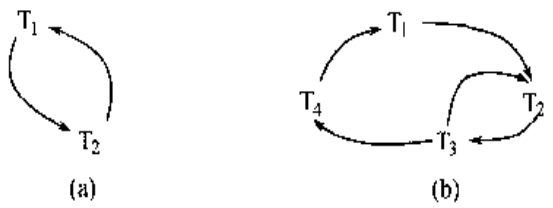


图 11.6 事务等待图

事务等待图动态地反映了所有事务的等待情况。并发控制子系统周期性地(比如每隔数秒)生成事务等待图,并进行检测。如果发现图中存在回路,则表示系统中出现了死锁。

图 11.6(a)表示事务 T_1 等待 T_2,T_2 等待 T_1,产生了死锁。图 11.6(b)表示事务 T_1 等待 T_2,T_2 等待 T_3,T_3 等待 T_4,T_4 又等待 T_1,产生了死锁。

当然,死锁的情况可以多种多样。例如,图 11.6(b)中事务 T_3 可能还等待 T_2,在大回路

中又有小的回路。这些情况人们都已经做了很深入的研究。

DBMS 的并发控制子系统一旦检测到系统中存在死锁，就要设法解除。通常采用的方法是选择一个处理死锁代价最小的事务，将其撤销，释放此事务持有的所有的锁，使其他事务得以继续运行下去。当然，对撤销的事务所执行的数据修改操作必须加以恢复。

11.4 并发调度的可串行性

DBMS 对并发事务不同的调度可能会产生不同的结果，那么什么样的调度是正确的呢？显然，串行调度是正确的。执行结果等价于串行调度的调度也是正确的。这样的调度叫做可串行化调度。

11.4.1 可串行化调度

定义 多个事务的并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同，称这种调度策略为**可串行化**(Serializable)的调度。

可串行性(Serializability)是并发事务正确调度的准则。按这个准则规定，一个给定的并发调度，当且仅当它是可串行化的，才认为是**正确调度**。

[例 3] 现在有两个事务，分别包含下列操作：

事务 T_1：读 B；A = B + 1；写回 A；

事务 T_2：读 A；B = A + 1；写回 B；

图 11.7 给出了对这两个事务不同的调度策略。

T_1	T_2
Slock B	
Y = R(B) = 2	
Unlock B	
Xlock A	
A = Y + 1 = 3	
W(A)	
Unlock A	
	Slock A
	X = R(A) = 3
	Unlock A
	Xlock B
	B = X + 1 = 4
	W(B)
	Unlock B

(a) 串行调度

T_1	T_2
	Slock A
	X = R(A) = 2
	Unlock A
	Xlock B
	B = X + 1 = 3
	W(B)
	Unlock B
Slock B	
Y = R(B) = 3	
Unlock B	
Xlock A	
A = Y + 1 = 4	
W(A)	
Unlock A	

(b) 串行调度

T_1	T_2
Slock B	
Y = R(B) = 2	
	Slock A
	X = R(A) = 2
Unlock B	
	Unlock A
Xlock A	
A = Y + 1 = 3	
W(A)	
	Xlock B
	B = X + 1 = 3
	W(B)
Unlock A	
	Unlock B

(c) 不可串行化的调度

T_1	T_2
Slock B	
Y = R(B) = 2	
Unlock B	
Xlock A	
	Slock A
A = Y + 1 = 3	等待
W(A)	等待
Unlock A	等待
	X = R(A) = 3
	Unlock A
	Xlock B
	B = X + 1 = 4
	W(B)
	Unlock B

(d) 可串行化的调度

图 11.7 并发事务的不同调度

假设 A、B 的初值均为 2。按 $T_1 \rightarrow T_2$ 次序执行结果为 A=3,B=4;按 $T_2 \rightarrow T_1$ 次序执行结果为 B=3,A=4。

图 11.7(a)和图 11.7(b)为两种不同的串行调度策略,虽然执行结果不同,但它们都是正确的调度;图 11.7(c)执行结果与(a)、(b)的结果都不同,所以是错误的调度;图 11.7(d)执行结果与串行调度(a)的执行结果相同,所以是正确的调度。

11.4.2 冲突可串行化调度

具有什么样性质的调度是可串行化调度呢?如何判断调度是可串行化调度呢?本节给出判断可串行化调度的**充分条件**。

首先介绍冲突操作的概念。

冲突操作是指不同的事务对同一个数据的读写操作和写写操作:

$R_i(x)$与 $W_j(x)$　　/* 事务 T_i 读 x,T_j 写 x */

$W_i(x)$与 $W_j(x)$　　/* 事务 T_i 写 x,T_j 写 x */

其他操作是不冲突操作。

不同事务的冲突操作和同一事务的两个操作是不能交换(Swap)的。对于 $R_i(x)$与 $W_j(x)$,若改变二者的次序,则事务 T_i 看到的数据库状态就发生了改变,自然会影响到事务 T_i 后面的行为。对于 $W_i(x)$与 $W_j(x)$,改变二者的次序,会影响数据库的状态,x 的值由等于 T_j 的结果变成了等于 T_i 的结果。

一个调度 Sc 在保证冲突操作的次序不变的情况下,通过交换两个事务不冲突操作的次序得到另一个调度 Sc',如果 Sc'是串行的,称调度 Sc 为**冲突**可串行化的调度。一个调度是冲突可串行化,一定是可串行化的调度。因此可以用这种方法来判断一个调度是否是冲突可串行化的。

[**例 4**] 今有调度 $Sc1 = r_1(A)w_1(A)r_2(A)\underline{w_2(A)}r_1(B)w_1(B)r_2(B)w_2(B)$

可以把 $w_2(A)$与 $r_1(B)w_1(B)$交换,得到:

$$r_1(A)w_1(A)\underline{r_2(A)}r_1(B)w_1(B)\underline{w_2(A)}r_2(B)\ w_2(B)$$

再把 $r_2(A)$与 $r_1(B)w_1(B)$交换:

$$Sc2 = r_1(A)w_1(A)r_1(B)w_1(B)\underline{r_2(A)}w_2(A)r_2(B)w_2(B)$$

Sc2 等价于一个串行调度 T_1,T_2。所以 Sc1 冲突可串行化的调度。

应该指出的是冲突可串行化调度是可串行化调度的**充分条件**,不是必要条件。还有不满足冲突可串行化条件的可串行化调度。

[**例 5**] 有 3 个事务 $T_1 = W_1(Y)W_1(X)$,$T_2 = W_2(Y)W_2(X)$,$T_3 = W_3(X)$

调度 $L_1 = W_1(Y)\underline{W_1(X)}W_2(Y)W_2(X)\ W_3(X)$是一个串行调度。

调度 $L_2 = W_1(Y)\underline{W_2(Y)}W_2(X)\underline{W_1(X)}W_3(X)$不满足冲突可串行化。但是调度 L_2 是可串行化的,因为 L_2 执行的结果与调度 L_1 相同,Y 的值都等于 T_2 的值,X 的值都等于 T_3

的值。

前面已经讲到，商用 DBMS 的并发控制--般采用封锁的方法来实现，那么如何使封锁机制能够产生可串行化调度呢？下面讲解的两段锁协议就可以实现可串行化调度。

11.5 两段锁协议

为了保证并发调度的正确性，DBMS 的并发控制机制必须提供一定的手段来保证调度是可串行化的。目前 DBMS 普遍采用两段锁(Two-Phase Locking，简称 2PL)协议的方法实现并发调度的可串行性，从而保证调度的正确性。

在运用封锁方法时，对数据对象加锁时需要约定一些规则，例如何时申请封锁、持锁时间、何时释放封锁等。我们称这些规则为**封锁协议**(Locking Protocol)。约定不同的规则，就形成了各种不同的封锁协议。两段封锁协议是最常用的一种封锁协议，理论上已经证明使用两段封锁协议产生的是可串行化调度。

所谓两段锁协议是指所有事务必须分两个阶段对数据项加锁和解锁。

- 在对任何数据进行读、写操作之前，首先要申请并获得对该数据的封锁；
- 在释放一个封锁之后，事务不再申请和获得任何其他封锁。

所谓“两段”锁的含义是，事务分为两个阶段，第一阶段是获得封锁，也称为扩展阶段。在这阶段，事务可以申请获得任何数据项上的任何类型的锁，但是不能释放任何锁。第二阶段是释放封锁，也称为收缩阶段。在这阶段，事务可以释放任何数据项上的任何类型的锁，但是不能再申请任何锁。

例如，事务 T_i 遵守两段锁协议，其封锁序列是：

Slock A　Slock B　Xlock C　Unlock B　Unlock A　Unlock C;

|—— 扩展阶段 ——| |—— 收缩阶段 ——|

又如，事务 T_j 不遵守两段锁协议，其封锁序列是：

Slock A　Unlock A　Slock B　Xlock C　Unlock C　Unlock B;

可以证明，若并发执行的所有事务均遵守两段锁协议，则对这些事务的任何并发调度策略都是可串行化的。

例如，图 11.8 的调度是遵守两段锁协议的，因此一定是一个可串行化调度。可以验证如下：忽略图中的加锁操作和解锁操作，按时间的先后次序得到了如下的调度：

$L_1 = R_1(A)R_2(C)W_1(A)W_2(C)R_1(B)W_1(B)R_2(A)W_2(A)$

通过交换两个不冲突操作的次序(先把 $R_2(C)$ 与 $W_1(A)$ 交换，再把 $R_1(B)W_1(B)$ 与 $R_2(C)W_2(C)$ 交换)，可得到：

$L_2 = R_1(A)W_1(A)R_1(B)W_1(B)R_2(C)W_2(C)R_2(A)W_2(A)$

因此 L_1 是一个可串行化调度。

需要说明的是，事务遵守两段锁协议是可串行化调度的**充分条件**，而不是必要条件。也就是说，若并发事务都遵守两段锁协议，则对这些事务的任何并发调度策略都是可串行化的；但是，若并发事务的一个调度是可串行化的，不一定所有事务都符合两段锁协议。例如图 11.7(d)是可串行化调度，但 T_1 和 T_2 不遵守两段锁协议。

另外要注意两段锁协议和防止死锁的一次封锁法的异同之处。一次封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行。因此一次封锁法遵守两段锁协议；但是两段锁协议并不要求事务必须一次将所有要使用的数据全部加锁，因此遵守两段锁协议的事务可能发生死锁，如图 11.9 所示。

事务 T_1	事务 T_2
Slock(A)	
R(A=260)	
	Slock(C)
	R(C=300)
Xlock(A)	
W(A=160)	
	Xlock(C)
	W(C=250)
	Slock(A)
Slock(B)	等待
R(B=1000)	等待
Xlock(B)	等待
W(B=1100)	等待
Unlock(A)	等待
	R(A=160)
	Xlock(A)
Unlock(B)	
	W(A=210)
	Unlock(C)

图 11.8 遵守两段锁协议的可串行化调度

T_1	T_2
Slock B	
R(B)=2	
	Slock A
	读 A=2
Xlock A	
等待	Xlock A
等待	等待

图 11.9 遵守两段锁协议的事务可能发生死锁

11.6 封锁的粒度

封锁对象的大小称为封锁粒度(Granularity)。封锁对象可以是逻辑单元，也可以是物理

单元。以关系数据库为例，封锁对象可以是这样一些逻辑单元：属性值、属性值的集合、元组、关系、索引项、整个索引直至整个数据库；也可以是这样一些物理单元：页（数据页或索引页）、物理记录等。

封锁粒度与系统的并发度和并发控制的开销密切相关。直观地看，封锁的粒度越大，数据库所能够封锁的数据单元就越少，并发度就越小，系统开销也越小；反之，封锁的粒度越小，并发度较高，但系统开销也就越大。

例如，若封锁粒度是数据页，事务 T_1 需要修改元组 L_1，则 T_1 必须对包含 L_1 的整个数据页 A 加锁。如果 T_1 对 A 加锁后事务 T_2 要修改 A 中元组 L_2，则 T_2 被迫等待，直到 T_1 释放 A。如果封锁粒度是元组，则 T_1 和 T_2 可以同时对 L_1 和 L_2 加锁，不需要互相等待，提高了系统的并行度。又如，事务 T 需要读取整个表，若封锁粒度是元组，T 必须对表中的每一个元组加锁，显然开销极大。

因此，如果在一个系统中同时支持多种封锁粒度供不同的事务选择是比较理想的，这种封锁方法称为**多粒度封锁**（Multiple Granularity Locking）。选择封锁粒度时应该同时考虑封锁开销和并发度两个因素，适当选择封锁粒度以求得最优的效果。一般说来，需要处理大量元组的事务可以以关系为封锁粒度；需要处理多个关系的大量元组的事务可以以数据库为封锁粒度；而对于一个处理少量元组的用户事务，以元组为封锁粒度就比较合适了。

11.6.1 多粒度封锁

下面讨论多粒度封锁，首先定义**多粒度树**。多粒度树的根结点是整个数据库，表示最大的数据粒度。叶结点表示最小的数据粒度。图 11.10 给出了一个三级粒度树。根结点为数据库，数据库的子结点为关系，关系的子结点为元组。也可以定义 4 级粒度树，例如数据库、数据分区、数据文件、数据记录。

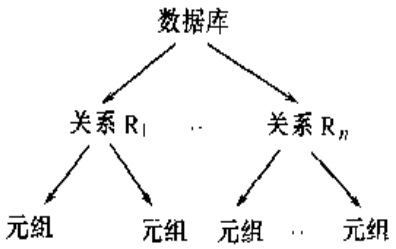


图 11.10 三级粒度树

然后，来讨论多粒度封锁的封锁协议。**多粒度封锁协议**允许多粒度树中的每个结点被独立地加锁。对一个结点加锁意味着这个结点的所有后裔结点也被加以同样类型的锁。因此，在多粒度封锁中一个数据对象可能以两种方式封锁，显式封锁和隐式封锁。

显式封锁是应事务的要求直接加到数据对象上的封锁；**隐式封锁**是该数据对象没有独立加锁，是由于其上级结点加锁而使该数据对象加上了锁。

多粒度封锁方法中，显式封锁和隐式封锁的效果是一样的，因此系统检查封锁冲突时不仅要检查显式封锁还要检查隐式封锁。例如事务 T 要对关系 R_1 加 X 锁。系统必须搜索其上级结点数据库、关系 R_1 以及 R_1 的下级结点，即 R_1 中的每一个元组，上下搜索。如果其中某一个数据对象已经加了不相容锁，则 T 必须等待。

一般地，对某个数据对象加锁，系统要检查该数据对象上有无显式封锁与之冲突；还要

检查其所有上级结点，看本事务的显式封锁是否与该数据对象上的隐式封锁（即由于上级结点已加的封锁造成的）冲突；还要检查其所有下级结点，看上面的显式封锁是否与本事务的隐式封锁（将加到下级结点的封锁）冲突。显然，这样的检查方法效率很低。为此人们引进了一种新型锁，称为**意向锁**（Intention Lock）。有了意向锁，DBMS 就无需逐个检查下一级结点的显式封锁。

11.6.2 意向锁

意向锁的含义是如果对一个结点加意向锁，则说明该结点的下层结点正在被加锁；对任一结点加锁时，必须先对它的上层结点加意向锁。

例如，对任一元组加锁时，必须先对它所在的数据库和关系加意向锁。

下面介绍三种常用的意向锁：意向共享锁（Intent Share Lock，简称 IS 锁）；意向排它锁（Intent Exclusive Lock，简称 IX 锁）；共享意向排它锁（Share Intent Exclusive Lock，简称 SIX 锁）。

1. IS 锁

如果对一个数据对象加 IS 锁，表示它的后裔结点拟（意向）加 S 锁。

例如，事务 T_1 要对 R_1 中某个元组加 S 锁，则要首先对关系 R_1 和数据库加 IS 锁。

2. IX 锁

如果对一个数据对象加 IX 锁，表示它的后裔结点拟（意向）加 X 锁。例如，事务 T_1 要对 R_1 中某个元组加 X 锁，则要首先对关系 R_1 和数据库加 IX 锁。

3. SIX 锁

如果对一个数据对象加 SIX 锁，表示对它加 S 锁，再加 IX 锁，即 SIX = S + IX。例如对某个表加 SIX 锁，则表示该事务要读整个表（所以要对该表加 S 锁），同时会更新个别元组（所以要对该表加 IX 锁）。

图 11.11(a)给出了这些锁的相容矩阵，从中可以发现这 5 种锁的强度如图 11.11(b)所示的偏序关系。所谓锁的强度是指它对其他锁的排斥程度。一个事务在申请封锁时以强锁代替弱锁是安全的，反之则不然。

具有意向锁的多粒度封锁方法中任意事务 T 要对一个数据对象加锁，必须先对它的上层结点加意向锁。申请封锁时应该按自上而下的次序进行；释放封锁时则应该按自下而上的次序进行。

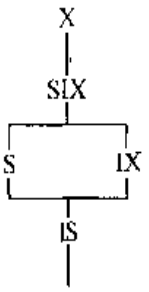
例如，事务 T_1 要对关系 R_1 加 S 锁，则要首先对数据库加 IS 锁。检查数据库和 R_1 是否已加了不相容的锁（X 或 IX）。不再需要搜索和检查 R_1 中的元组是否加了不相容的锁（X 锁）。

具有意向锁的多粒度封锁方法提高了系统的并发度，减少了加锁和解锁的开销，它已经在实际的数据库管理系统产品中得到广泛应用。

T_1 \ T_2	S	X	IS	IX	SIX	-
S	Y	N	Y	N	N	Y
X	N	N	N	N	N	Y
IS	Y	N	Y	Y	Y	Y
IX	N	N	Y	Y	N	Y
SIX	N	N	Y	N	N	Y
-	Y	Y	Y	Y	Y	Y

Y=Yes，表示相容的请求　　N=No，表示不相容的请求

(a) 数据锁的相容矩阵



(b) 锁的强度的偏序关系

图 11.11　加上意向锁后锁的相容矩阵与偏序关系

11.7 小　结

数据库的重要特征是它能为多个用户提供数据共享。数据库管理系统允许共享的用户数目是数据库管理系统重要标志之一。数据库管理系统必须提供并发控制机制来协调并发用户的并发操作以保证并发事务的隔离性和一致性，保证数据库的一致性。

数据库的并发控制以事务为单位，通常使用封锁技术实现并发控制。本章介绍了最常用的封锁方法。对数据对象施加封锁，会带来活锁和死锁问题，并发控制机制必须提供适合数据库特点的解决方法。

并发控制机制调度并发事务操作是否正确的判别准则是可串行性，两段锁协议是可串行化调度的充分条件，但不是必要条件。因此，两段锁协议可以保证并发事务调度的正确性。

不同的数据库管理系统提供的封锁类型、封锁协议、达到的系统一致性级别不尽相同。但是其依据的基本原理和技术是共同的。

习　题

1. 在数据库中为什么要并发控制？并发控制技术能保证事务的哪些特性？
2. 并发操作可能会产生哪几类数据不一致？用什么方法能避免各种不一致的情况？
3. 什么是封锁？基本的封锁类型有几种？试述它们的含义。
4. 如何用封锁机制保证数据的一致性？
5. 什么是活锁？试述活锁的产生原因和解决方法。
6. 什么是死锁？请给出预防死锁的若干方法。
7. 请给出检测死锁发生的一种方法，当发生死锁后如何解除死锁？

8. 什么样的并发调度是正确的调度？

9. 设T_1、T_2、T_3是如下的3个事务，设A的初值为0；

T_1：A：=A+2；

T_2：A：=A*2；

T_3：A：=A**2；（A←A^2）

若这3个事务允许并发执行，则有多少种可能的正确结果，请一一列举出来；

请给出一个可串行化的调度，并给出执行结果；

请给出一个非串行化的调度，并给出执行结果；

若这3个事务都遵守两段锁协议，请给出一个不产生死锁的可串行化调度；

若这3个事务都遵守两段锁协议，请给出一个产生死锁的调度。

10. 今有3个事务的一个调度$r_1(B)r_1(A)w_3(B)r_2(B)r_2(A)w_2(B)r_1(B)w_1(A)$，该调度是冲突可串行化的调度吗？为什么？

11. 试证明，若并发事务遵守两段锁协议，则对这些事务的并发调度是可串行化的。

12. 举例说明，对并发事务的一个调度是可串行化的，而这些并发事务不一定遵守两段锁协议。

13. 为什么要引进意向锁？意向锁的含义是什么？

14. 试述常用的意向锁：IS锁、IX锁、SIX锁，给出这些锁的相容矩阵。

15. 理解并解释下列术语的含义：封锁、活锁、死锁、排它锁、共享锁、并发事务的调度、可串行化的调度、冲突可串行化调度、两段锁协议。

*16. 试述你了解的某一个实际的DBMS产品的并发控制机制。

本章参考文献

1 Bernstein P A. Concurrency Control and Recovery In Database Systems. 1986

2 Eswaran K P, Gray J N, Lorie R A, et al. The Notions of Consistency and Predicate Locks in a Data Base System. CACM 19, No. 11, 1976

（本文给出了可串行性概念的形式化定义。）

3 Gray J N, Lorie R A, Putzolu G R. Granularity of Locks in a Large Shared Data Base. in Proceedings of VLDB, 1975

4 Papadimitriou C H, Bernstein P A, Rothnie J B. Some Computational Problems Related to Database Concurrency Control. Proceedings of the Conference on Theoretical Compute Science, 1977

5 Papadimitriou C H. The Serializability of Concurrent Database Updates. Journal of the ACM, Volume 26, Number 4, 1979

（本文给出了与可串行性相关的研究结果。）

6 Soisalon-Soininen E, Wood D. An Optimal Algorithm for Testing Safety and Detecting Deadlocks. Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 1982

7 Yannakakis M, Papadimitriou C H, Kung H T. Locking Protocols: Safety and Freedom from Deadlock. Proceedings of the IEEE Symposium on the Foundations of Computer Science, 1979

（本文讨论了封锁协议。）

8 Kedem Z M, Silberschatz A. Locking Protocols: From Exclusive to Shared Locks. Journal of the ACM, Volume 30, Number 4, 1983

（本文讨论封锁协议。）

9 Korth H F. Locking Primitives in a Database System. Journal of the ACM, Volume 30, Number 1, 1983

（本文探讨了从基本共享锁和独占锁模型获得的各种锁模型。）

10 Papadimitriou C H. A Theorem in Database Concurrency Control. Journal of the ACM, Volume 29, Number 5, 1982

11 Bayer R, Schkolnick M. Concurrency of Operating on B-trees. Acys Informatica, Volume 9, Number 1, 1977

（本文提出了对 B－树进行并发存取的算法。）

12 Ellis C S. Concurrent Search And Insertion in 2－3 Trees. Acta Informatica, Volume 14, 1980

（本文提出了对 2－3 树进行并发存取的算法。）

13 Ellis C S. Concurrent Search And Insertion in AVL Trees. IEEE Transactions on Computers, Volume E－29, Number 3, 1980

（本文提出了对 AVL 树进行并发存取的算法。）

14 Lehman P L, Yao S B. Efficient Locking for Concurrent Operations on B-trees. ACM Transactions on Database Systems, Volume 6, Number 4, 1981

15 Manber U, Ladner R E. Concurrency Control in a Dynamic Search Structure. ACM Transactions on Database Systems, Volume 9, Number 3, 1984

（本文讨论了对动态搜索树进行并发存取的各种算法。）

16 Buckley G, Silberschatz A. Concurrency Control in Graph Protocols by Using Edge Locks. Proceedings of the ACM SIGACT-SIGMOD Symposium on the Principles of Database Systems, 1984

（本文给出了扩展的封锁协议。）

17 Bernstein P A, Goodman N. Timestamp-based Algorithms for Concurrency Control in Distributed Database Systems. Proceedings of the International Conference on Very Lorge Data Bases, 1980

（本文讨论了各种基于时间戳的并发控制算法。）

18 Gray J N, Lorie R A, Putzolu G R. Granularity of Locks and Degrees of Consistency in a Shared Data Base. in Proceedings of VLDB, 1975

（本文综述了多粒度封锁协议。）

* 第十二章 数据库管理系统

在第一章中，已简单介绍了数据库管理系统(DBMS)的功能。本章进一步阐述 DBMS 的基本功能、系统结构及主要的实现技术。本章不是针对 DBMS 设计人员写的，而是针对 DBA 和数据库应用系统开发人员的，目的是使他们从宏观和总体的角度掌握 DBMS 的基本概念和基本原理，以便更好地使用和维护 DBMS。

12.1 DBMS 的基本功能

数据库管理系统已经发展成为继操作系统之后最复杂的系统软件。前面已讲过，DBMS 主要是实现对共享数据有效的组织、存储、管理和存取。围绕数据，DBMS 应具有如下基本功能。

1. 数据库定义和创建

创建数据库主要是用数据定义语言 DDL 定义和创建数据库模式、外模式、内模式等数据库对象。在关系数据库中就是建立数据库(或 Schema)、表、视图、索引等。还有创建用户、安全保密定义(如用户口令、级别、角色、存取权限)、数据库的完整性定义。这些定义存储在数据字典(亦称为系统目录)中，是 DBMS 运行的基本依据。

2. 数据组织、存储和管理

DBMS 要分类组织、存储和管理各种数据，包括数据字典、用户数据、存取路径等。要确定以何种文件结构和存取方式在存储器上组织这些数据，如何实现数据之间的联系。数据组织和存储的基本目标是提高存储空间利用率和方便存取，提供多种存取方法(如索引查找、Hash 查找、顺序查找等)提高存取效率。

3. 数据存取

提供用户对数据的操作功能，实现对数据库数据的检索、插入、修改和删除。一个好的 RDBMS 应该提供功能强易学易用的数据操纵语言(DML)、方便的操作方式和较高的数据存取效率。DML 有两类：一类是宿主型语言，一类是自立(独立)型语言。

4. 数据库事务管理和运行管理

这是指 DBMS 运行控制和管理功能。包括多用户环境下的事务管理功能和安全性完整

性控制功能。包括数据库恢复、并发控制和死锁检测(或死锁防止)、安全性检查和存取控制、完整性检查和执行、运行日志的组织管理等。这些功能保证了数据库系统的正常运行,保证了事务的 ACID 特性。

5. 数据库的建立和维护

包括数据库的初始建立、数据的转换、数据库的转储和恢复、数据库的重组织和重构造以及性能监测分析等功能。

6. 其他功能

包括 DBMS 与网络中其他软件系统的通信功能;一个 DBMS 与另一个 DBMS 或文件系统的数据转换功能;异构数据库之间的互访和互操作功能等。此外,随着技术的发展,许多新的应用对 DBMS 提出了新的需求。DBMS 要不断发展新的数据管理技术,例如 XML 数据、流数据、空间数据、多媒体数据等。

和操作系统、编译系统等系统软件相比,DBMS 具有跨度大、功能多的特点。从最底层的存储管理、缓冲区管理、数据存取操作、语言处理到最外层的用户接口、数据表示、并发环境的支持都是它要实现的功能。

DBMS 的实现,既要充分利用计算机硬件、操作系统、编译系统和网络通信等技术,又要突出对海量数据存储、管理和处理的特点,并要保证其存取数据和运行事务的高效率,这是一个复杂而综合的软件设计开发过程。在下面几节,将逐步深入地讨论 DBMS 的进程结构、基本功能及语言处理等实现方法和技术。

12.2 DBMS 的进程结构和多线索机制

实际应用中,DBMS 以及应用程序都必须进入某个具体的操作系统环境。应用程序通过 DBMS,DBMS 通过 OS,与数据库中的数据打交道。根据三者的关系,一般有下述四种进程组织方案。

12.2.1 N 方案:DBMS 与应用程序相融合的方案

在这种方案中,N 个 DB 用户仅需 N 个进程,DBMS 作为应用程序的子程序被连入应用程序中,成为应用程序的一部分,因此这种方案也称为连入式方案,如图 12.1 所示。

其中 AP_i 为数据库系统应用程序;U_i 为非数据库用户(或应用程序);SGA 为共享全局区(Shared Global Area),存放字典定义信息、数据和索引的缓冲块、日志缓冲块以及封锁控制块等全局资源信息。

N 方案最明显的特点是能用子程序调用方式把控制迅速直接地传送给 DBMS,从而减少应用程序的运行时间。与后面的几种方案相比,N 方案调用一次 DBMS 至少省掉两次进程切换开销。而在大多数操作系统中进程切换通常是很费时间的,大约需要数千条指令。

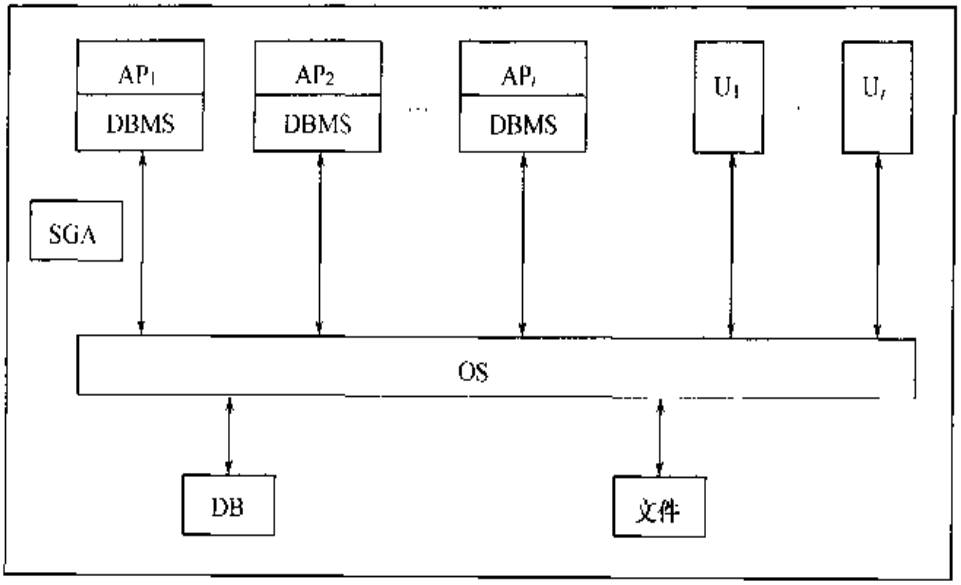


图 12.1 N 方案

作为一种子程序技术，N 方案对内存的需求量比较大。因为用户程序与 DBMS 功能的融合使得 DBMS 的代码段无法共享，导致内存中保留了 DBMS 代码的许多副本。随着 DBMS 功能的日益增强，DBMS 代码会不断扩大，N 方案造成的代码冗余会使系统性能下降，甚至达到令人无法忍受的程度。

N 方案 DBMS 实现起来比较简单，不用考虑用户进程与 DBMS 进程间的通信，因此在用户数少的小型 DBMS 中可以采用该方法。

解决 N 方案 DBMS 代码冗余的办法是将图 12.1 中的 AP_i 与其 DBMS 副本分开，让 DBMS 的代码段在内存中被共享，这就引出了下面的方案。

12.2.2 2N 方案：一个 DBMS 进程对应一个用户进程

每个用户进程均有一个 DBMS 进程为之服务，称这种 DBMS 进程为影子(Shadow)进程。在这种方案中，有 N 个用户进程就要启动 N 个 DBMS 进程(共 2N 个进程)，因此这种进程结构方案称为 2N 方案，如图 12.2 所示。

在这种情况下，数据库系统的各个活跃进程(Shadow 进程)都是独立运行的。因此，用户进程与 Shadow 进程之间以及各 Shadow 进程相互间都要通信。

系统的全局管理数据驻留在 SGA 中，各 DBMS 进程靠操作系统提供的封锁原语(信号量)实现同步或实现对 SGA 的互斥访问。因此 DBMS 的这种进程组织方式使进程间总的通信开销上升很多。

在 2N 结构的方案中，操作系统的负担增大了，主要表现在空间和时间两个方面。在空间上，虽然各个 DBMS 进程的代码段(如 UNIX 中的正文段)可以共享，但数据段和栈段的空间还是各自的，再加上 OS 要为多出来的 N 个 Shadow 进程分配进程控制块(PCB)等诸多内

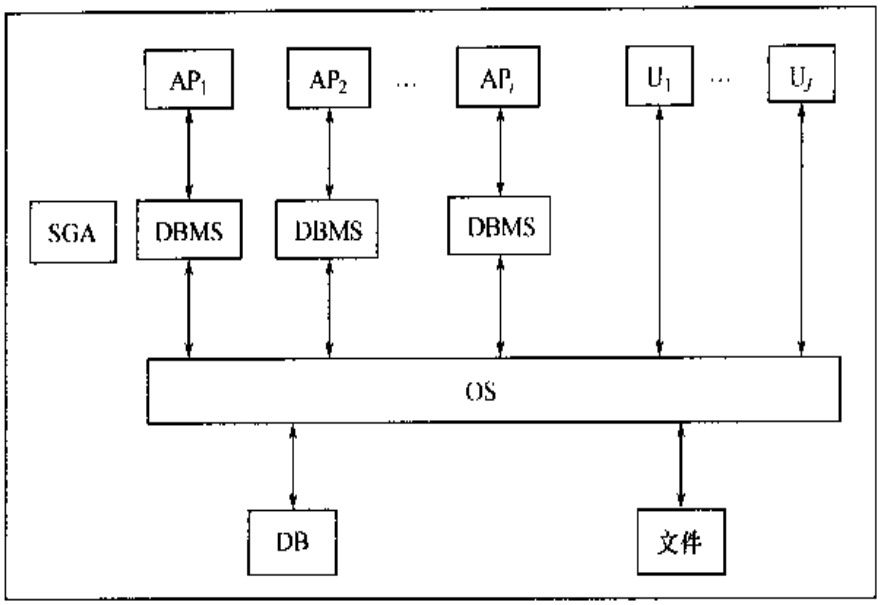


图 12.2 2N 方案

部空间,因此 2N 结构对内存的需要量亦很大。这极易使内存页面状况变坏,并可能因物理内存远远小于各进程虚空间之和而导致各进程被操作系统频繁地换入换出(Swap in/Swap out),使系统出现颠簸。在时间方面,由于 2N 方案中进程数目过多,许多 CPU 时间会白白浪费在进程切换这种昂贵的工作上。颠簸及进程切换的高额开销使系统性能严重下降。

2N 方案中增加一个用户所需要的资源很大,通常增加两三个用户就需要 1MB 的内存量,再加上操作系统所支持的进程总数有限,导致了 2N 方案不可能支持很多的 DBMS 用户,因此它不大适合于大量用户的 OLTP 应用。

除了每个用户进程需要一个 DBMS 服务进程外,DBMS 在全局上还要维护若干个后台服务进程。由这些后台进程通过 SGA 完成诸如监控、写数据库、预读、写日志、淘汰页面、死锁检测和解除等全局工作。使用后台进程大大增加了进程间通信的开销,而频繁的进程通信意味着频繁的进程切换。少数几个后台进程还可能成为整个系统的瓶颈。

此外,2N 方案至少还会碰到数据不在内存时所造成的性能问题和临界区(Critical Section)问题,这里就不再详细讨论了。

把多个 DBMS 进程合为一个 DBMS 进程可以有效地解决上述问题,这就是 N+1 方案。与 N+1 方案相比较,2N 方案的最大优势是它省掉了 DBMS 对各数据库用户的多任务调度,而把这个任务交给了操作系统,从而简化了用户进程与 DBMS 的接口。

Oracle 7 之前的版本、Ingres 和 Informix 的早期版本使用 2N 方案(最早的 INGRES 采用 4N 方案:每个用户进程有 3 个分别完成不同任务的 DBMS 进程为之服务)。

12.2.3 N+1 方案:一个 DBMS 进程对应所有用户进程

进程结构的另一种组织方式是整个 DBMS 仅使用一个进程,该进程的行为类似于一个

服务器(Server)。多个数据库用户向 Server 发消息,以申请数据库服务。Server 要用自己的机制来调度这些申请,以支持一个多任务的数据库系统。由于 N 个用户仅需要一个 DBMS 进程,因此称这种进程结构方案为 $N+1$ 方案,如图 12.3 所示,这时 SGA 已不再需要了。

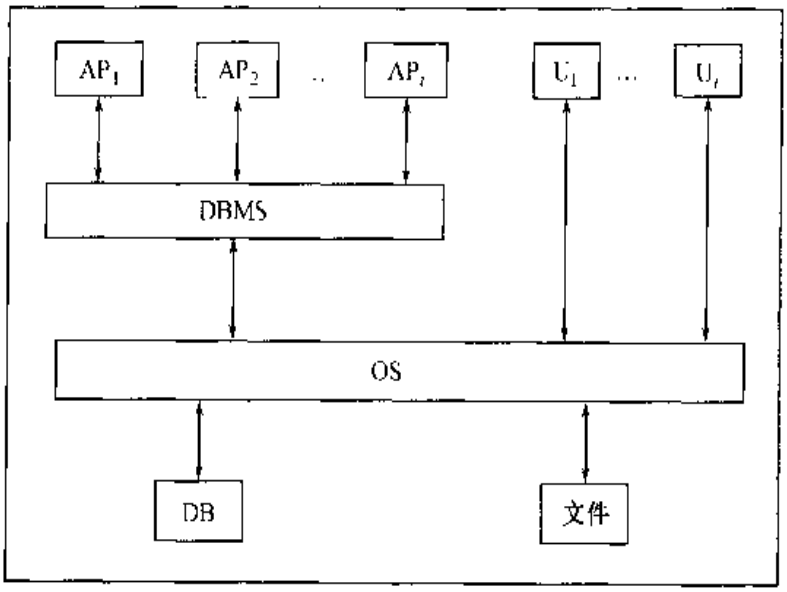


图 12.3　$N+1$ 方案

由于要设计自己的多任务处理机制和调度算法,使得 $N+1$ 方案的 DBMS 设计在整体上比 $2N$ 方案要复杂。Server 要处理所有用户的申请,如果调度策略不当或不能使数据库进程获得较高的优先权,这种方案将导致瓶颈。

另外,$N+1$ 方案中要用到操作系统级的消息机制以实现多进程向单进程以及单进程向多进程的数据传送。许多文献指出消息系统是一种昂贵的设施,大多数操作系统中消息往返一次的开销是几千条指令。

尽管如此,$N+1$ 方案的许多诱人的优点仍使许多 RDBMS 厂家纷纷朝这个方向靠拢。

这种方案的另一个优点是可以采用多线索(Multi_Threaded)技术来实现 $N+1$ 方案,从而可大大提高系统性能,降低系统资源的开销,简化 DBMS 许多部分的设计。这是因为线索机制使得 $N+1$ 方案能用统一的"线索"思想处理所有执行流,其中包括 DBMS 核心的执行流,这将省去所有后台进程并能适应系统软件设计与并发的微内核(Micro_Kernel)要求,使 DBMS 的核心代码简明精巧、易于移植。在 12.2.5 小节中将讲解多线索的概念和技术。

12.2.4　$N+M$ 方案:M 个 DBMS 进程对应 N 个用户进程

$N+M$ 方案采用 M 个 DBMS 进程为 N 个用户进程提供服务(一般 $M<N$),如图 12.4 所示。

在该方案中,DBMS 进程不负责多任务调度,同时每个用户进程也不固定地对应于某个 DBMS 进程。用户的数据库请求将会被动态地分配给某个 DBMS 进程来处理(例如最不忙

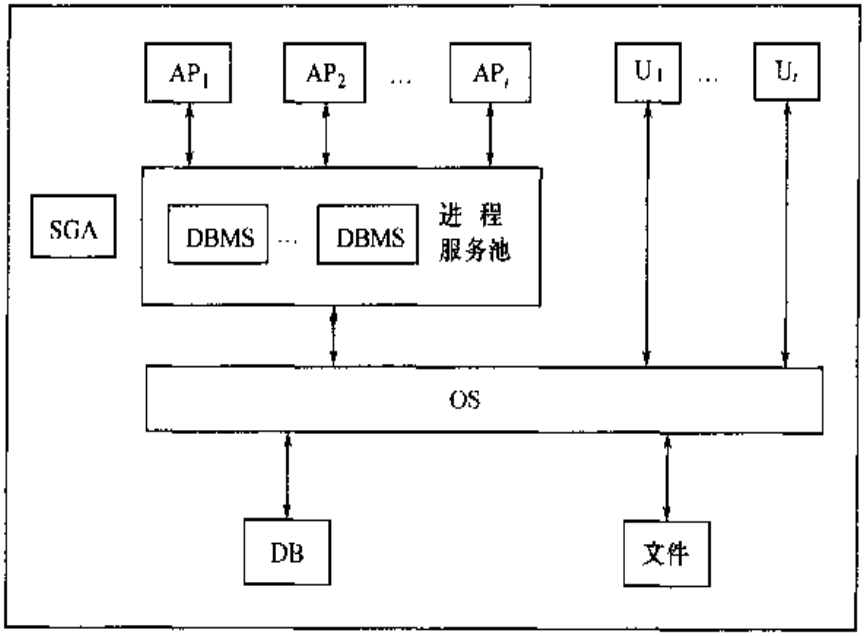


图 12.4 $N+M$ 方案

的那个 DBMS 进程)。

DBMS 进程的分派由分派程序完成。分派程序可以是一个单独的后台进程,也可以将其代码连入用户程序(AP_i)中。分派程序通过 SGA 管理用户的数据库请求队列,并分配某个 DBMS 进程服务于某用户进程。分派程序还应监测整个 DBMS 的运行状况并根据用户请求队列的情况动态增减 DBMS 进程的个数。如果用户进程增加,则 DBMS 进程的个数也动态地增多,但后者一般总小于前者(即 $M<N$)。

$N+M$ 方案基于的指导思想是:用户进程可能并不总是同时提出数据库请求,一个用户一个 DBMS 进程的方案太浪费了,几个用户轮流使用一个 DBMS 服务进程也许会节约一些。

$N+M$ 方案是 $2N$ 方案的一种改进,提高了内存资源的利用率,但它没有克服 $2N$ 方案的本质弱点。此外,分派程序给系统增加了开销并可能成为瓶颈,而且 DBMS 进程动态增减的开销亦很大。

12.2.5 多线索(Multi_Threaded)DBMS 的概念

一、线程的概念

数据库系统中线索(Thread)的概念来源于操作系统中"线程"(Thread)的概念。这里先介绍操作系统中线程的概念。

随着多处理器系统以及并行计算技术的发展,进程概念被进一步细划为"任务"(Task)与"线程"的概念。

UNIX 核心中最关键的概念就是进程。进程既是资源分配的最小单位也是运行的最小单位。与一个进程相联系的资源有:进程虚拟空间、进程控制块、数据段、正文段、堆栈段、共

享内存、程序计数器、状态寄存器及通用寄存器等。进程建立及切换的开销很大，核心维护每一个进程的虚拟空间开销也很大。进程概念在处理微粒度问题（一个问题可划分成多个工作量很小的并行计算单位）时显得很吃力，不能充分发挥多机的优点。出于以上考虑，1988 年卡内基－梅隆大学在其研制的多机操作系统 MACH 中实现了 Task 及 Thread 这两个概念。

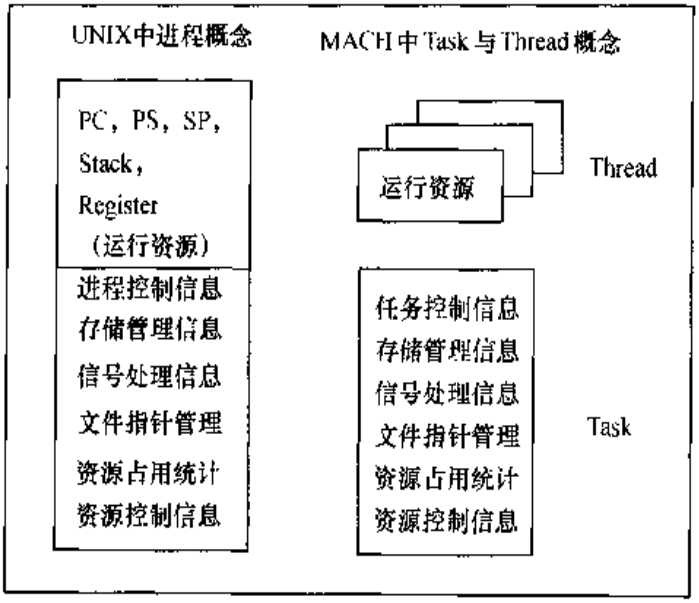


图 12.5　进程、Task 与 Thread

他们将进程的概念一分为二：**Task** 是申请资源的最小单位，而 **Thread** 是调度和运行的最小单位。一个 Task 中可有多个 Thread，这些 Thread 共享 Task 的所有资源，共同完成一个任务。一个 Thread 只能存在于某个 Task 中。在多处理机上，这些 Thread 可真正地并行执行。UNIX 中的进程概念相当于 MACH 中的只包括一个 Thread 的 Task。图 12.5 说明了它们之间的区别。

从图 12.5 中可以看出，由于多个线程运行于同一实体（任务或进程）中并且共享该实体所拥有的所有资源，使得传统 UNIX 中属于某一任务的多个进程（特别是个数较多情况下）以线程的方式并行执行时效率大大提高（所有线程共享一个地址空间等）。线程之间的切换效率更高，因此线程又被称为“轻权”或“轻量”进程（light_weighted process）。

可见，线程的基本概念就是将进程中的程序代码与进程所占资源相分离，从而在一个地址空间可以运行多个指令流。线程概念在表达应用问题本身的并行性方面较传统进程有较大优势，因而可以更好地发挥多处理机系统的能力。

二、多线索（Multi_Threaded）DBMS

数据库系统中的“线索”概念借鉴了操作系统中“线程”概念的含义：整个 DBMS 可以看作是一个 Task，当有一个用户申请数据库服务时，Task 分配至少一个 Thread 为之服务，多个 Thread 并行工作，共享资源。

一般地讲，DBMS 中的线索是 DBMS 的一个执行流，它服务于整个 DBMS 系统或 DBMS 中的某个用户；DBMS 服务器响应客户请求是通过为每个用户创建线索（而不是创建进程）来完成的。DBMS 的各个线索能在逻辑上并行执行；它们共存于一个进程中，共享 DBMS 的所有资源，如数据库缓冲区和 CPU 时间；线索是 DBMS 的调度单位，服务器进程能按一定的调度算法调度用户请求，由于调度优化策略是由数据库服务器执行，因此会比操作系统直接

对这些请求进行调度高效得多。

多线索机制可以减少每个用户需要的系统资源，如内存要求，从而可以增加并发用户数，提高服务器的运行效率，更好地支持 OLTP 应用。

在支持线程的操作系统（如 MACH、Chorus、Amoeba 等）上实现单进程多线索 DBMS 是比较容易的，这种操作系统能给予 DBMS 较好的支持。

三、线索与进程的比较

为了进一步说明线索概念对于 DBMS 设计和实现的重要意义，下面对线索机制与进程机制作一个比较。

1. 线索比进程占用较少的资源

进程是程序的动态概念，指正在执行的程序。进程有其自身的程序地址空间，除正文段以外，进程间一般不能共享彼此资源。线索则是程序中的一串指令流，同一进程内的多个线索共享该进程内的一切资源。因此，线索比进程占用更少的系统资源，是更小的调度单位，可支持较细粒度的并行。

2. 线索调度比较灵活，可控制性强

线索的调度方式比较灵活，可以实现“智能”化的调度，提高并行效率。例如，可以让紧急事务优先执行，可以设置线索优先级，可以使某一线索主动出让 CPU 等。

进程调度由于是由操作系统控制的，DBMS 无法干涉，而且进程调度是非“智能”的，特定进程可能会被频繁挂起。例如，假定一个进程要反复地读取数据、处理数据并回写数据，这样该进程会因等待 I/O 而被频频挂起，效率很低；但如果将该进程划分为读、处理、写等 3 个线索，这样当其中某个线索被阻塞时，其他线索可被调度执行，效率就会提高。而且，可以通过合理地调度一个进程内的多条并发线索，而使该进程长久地占用 CPU 。从这一意义上讲，无论单处理机还是多处理机，多线索结构均能提高执行效率。

3. 线索切换开销较小

就进程切换而言，由于进程的地址空间是私有的，因此进程切换必须保持进程的正文、数据、栈段及地址映射、进程状态、寄存器值等系统信息，一般需要花费数千条机器指令；而线索由于共享同一地址空间，因此当切换同一进程内的不同线索时，只需保存线索状态、寄存器值（程序计数器、堆栈指针、通用寄存器）等，由于切换较少的信息，线索切换的代码量明显减少，一般需数十条指令。

4. 线索间通信简便

UNIX 操作系统将进程的程序地址空间相隔离是为了避免并发进程间的相互干扰，以后为了支持进程间通信又先后引入了管道、通道、新 IPC 机制（信号量、消息及共享存储器）等通信手段。这些通信机制或者编程麻烦、不易使用，或者有一定局限性、效率不高。同一进程内的线索由于共享地址空间，因而可以借助全局变量名或指向局部变量名的指针来进行通信，这种通信当然简单而直接。不过，由于线索间可以互访数据，因此在利用线索编程时

应注意保持数据的一致性和完整性。

总之，采用多线索机制的 DBMS 具有运行效率高、消耗系统资源少等显著优点。

12.3　DBMS 系统结构

12.3.1　DBMS 的层次结构

和操作系统一样，可以将 DBMS 划分成若干层次。清晰、合理的层次结构不仅可以更清楚地认识 DBMS，更重要的是大大有助于 DBMS 的设计和维护。

例如 IBM 公司最早研制的著名的 RDBMS 实验系统 SystemR，其核心分为两层，即底层的关系存储系统 RSS 和上层的关系数据系统 RDS。RDS 本质上是一个语言和执行层，完成包括语法检查与分析、优化、代码生成、视图实现、安全性完整性检查等功能。RSS 则是一个存取方法层，其功能包括空间和设备管理、索引和存取路径管理、事务管理、并发控制、运行日志管理和恢复等。

图 12.6 给出一个 RDBMS 的层次结构示例。这个层次结构是按照处理对象的不同，依最高级到最低级的次序来划分的，具有普遍性。图中包括了与 RDBMS 密切相关的应用层和操作系统。

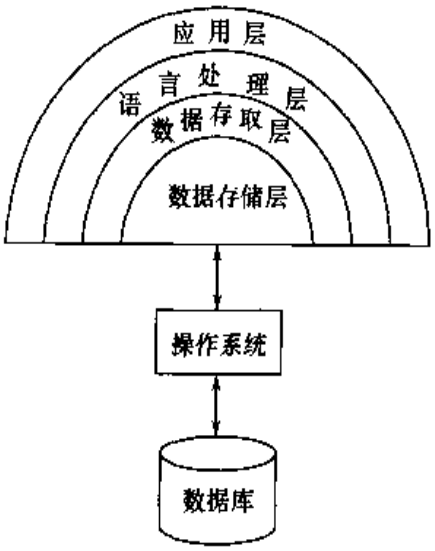


图 12.6　RDBMS 的层次结构

最上层是应用层，位于 RDBMS 核心之外。它处理的对象是各种各样的数据库应用，如用开发工具开发的或用 ESQL、存储过程等编写的应用程序、终端用户通过应用接口发出的事务请求或各种查询要求等。该层是 RDBMS 与用户/应用程序的界面层。

第二层是语言处理层。它处理的对象是数据库语言，如 SQL。向上提供的数据接口是关系、视图，即元组的集合。其功能是对数据库语言的各类语句进行语法分析、视图转换、安全性检查、完整性检查、查询优化等。通过对下层基本模块的调用，生成可执行代码。这些代码的运行即可完成数据库语句的功能要求。

第三层是数据存取层。该层处理的对象是单个元组。它把上层的集合操作转换为单记录操作。执行扫描（如表扫描）、排序、元组的查找、插入、修改、删除、封锁等基本操作。完成数据记录的存取、存取路径维护、事务管理、并发控制和恢复等工作。

第四层是数据存储层。该层处理的对象是数据页和系统缓冲区。执行文件的逻辑打开、关闭、读页、写页、缓冲区读和写、页面淘汰等操作，完成缓冲区管理、内外存交换、外存的数据管理等功能。

操作系统是 RDBMS 的基础。它处理的对象是数据文件的物理块。执行物理文件的读写操作,保证 RDBMS 对数据逻辑上的读写真实地映射到物理文件上。操作系统提供的存取原语和基本的存取方法通常作为和 RDBMS 存储层的接口。

以上所述的 RDBMS 层次结构划分的思想具有普遍性。当然具体系统在划分细节上会是多种多样的,这可以根据 RDBMS 实现的环境、系统的规模灵活地处理。

下面将按照由外及内的次序在下面 12.4 ~ 12.7 节中依次介绍图 12.6 中的各个层次。

12.3.2 RDBMS 的运行过程示例

RDBMS 是一个复杂而有序的整体,应该用动态的观点看待 RDBMS 各个功能模块。下面考察一个应用程序/用户通过 RDBMS 读取数据库中数据的过程(图 12.7),来加深对 RDBMS 的了解。图中的数据字典是数据库的重要组成部分,存储数据库定义。

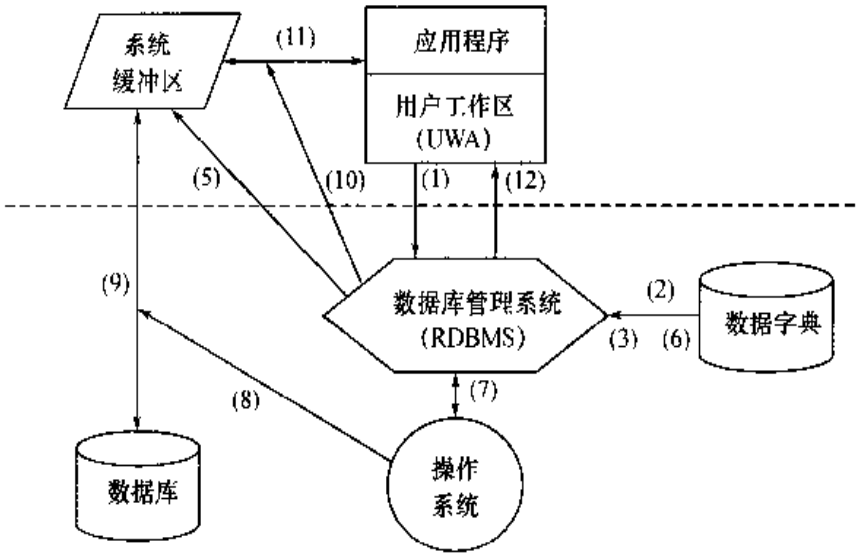


图 12.7 RDBMS 的运行过程示例

(1) 用户 A 通过应用程序向 RDBMS 发出调用数据库数据的命令,如 SELECT 命令。命令中给出了一个关系名和查找条件。

(2) RDBMS 先对命令进行语法检查。语法检查通过后进行语义检查和用户存取权限检查。具体做法是,RDBMS 读取数据字典,检查是否存在该关系及相应的字段,该用户能否读取它们等。确认语义正确、存取权限合法后便决定执行该命令,否则拒绝执行,返回错误信息。

(3) RDBMS 执行查询优化。优化器要依据数据字典中的信息进行优化,并把该命令转换成一串单记录的存取操作序列。

(4) RDBMS 执行存取操作序列(反复执行以下各步,直至结束)。

(5) RDBMS 首先在缓冲区中查找记录,若找到满足条件的记录则转到(10),否则转到(6)。

(6) RDBMS 查看存储模式,决定从哪个文件、用什么方式读取哪个物理记录。

(7) RDBMS 根据(6)的结果,向操作系统发出读取记录的命令。

(8) 操作系统执行读数据的有关操作。

(9) 操作系统将数据从数据库的存储区送至系统缓冲区。

(10) RDBMS 根据查询命令和数据字典的内容导出用户所要读取的记录格式。

(11) RDBMS 将数据记录从系统缓冲区传送到应用程序 A 的用户工作区。

(12) RDBMS 将执行状态信息,如成功读取或不成功的错误指示、例外状态信息等返回给应用程序 A。

对照在 12.3.1 节中给出的 RDBMS 层次结构,可以大致作如下的对应:

动作(1)属于第一层——应用层。

动作(2)、(3)由第二层——语言处理层来完成。

动作(4)、(10)、(11)、(12)由第三层——数据存取层来完成。

动作(5)、(6)、(7)由第四层——数据存储层来进行。

动作(8)、(9)由操作系统执行。

整个 RDBMS 各层模块互相配合、互相依赖共同完成对数据库的操纵。

对其他一些操作,如插入、删除、修改,其过程与读一个记录是类似的。

12.4 语言处理

前面已经介绍,RDBMS 一般向用户提供多种形式的语言,如交互式命令语言(如 SQL)、嵌入主语言的嵌入式语言(如 ESQL)、过程化语言(如 PL/SQL 和存储过程)。这些语言都是由 RDBMS 的语言处理层来支持的。

12.4.1 语言处理层的任务和工作步骤

语言处理层的任务就是把用户在这各种方式下提交给 RDBMS 的数据库语句转换成对 RDBMS 内层可执行的基本存取模块的调用序列。

数据库语言通常包括 DDL、DML、DCL 三部分语句。DDL 语句处理相对独立和简单;DML 和 DCL 则较为复杂。

具体来说,对 DDL 语句,语言处理层首先把它翻译成内部表示,然后把它存储在系统的数据字典中。对 DCL 语句的定义部分,如安全保密定义、存取权限定义、完整性约束定义等的处理与 DDL 相同。

在 RDBMS 中数据字典通常采用和普通数据同样的表示方式,即也用关系表(Table)来表示。数据字典包括关系定义表、属性表、视图表、视图属性表、视图表达式表、用户表、用户存取权限表等。图 12.8 给出了一个 RDBMS 数据字典的一部分示意图。

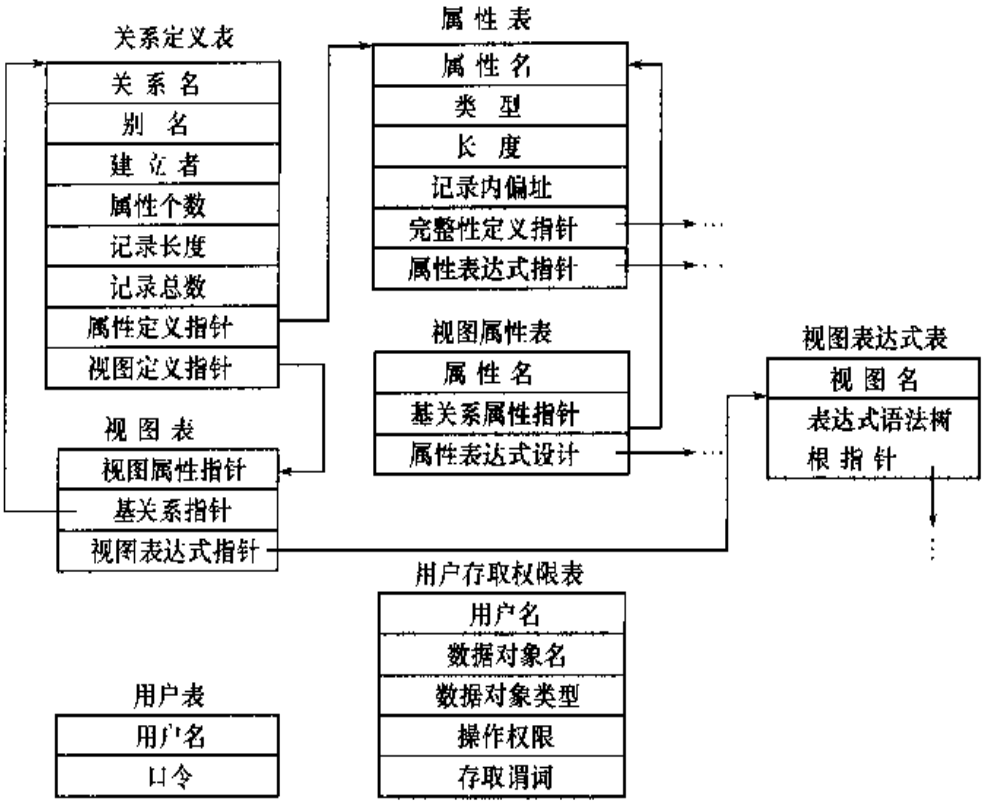


图 12.8 RDBMS 数据字典的部分示意图

数据字典是 DML 语句处理、执行以及 RDBMS 运行管理的基本依据。

对 DML 语句,语言处理层要做的工作比较多,图 12.9 给出了 RDBMS 中 DML 语句处理过程的示意。

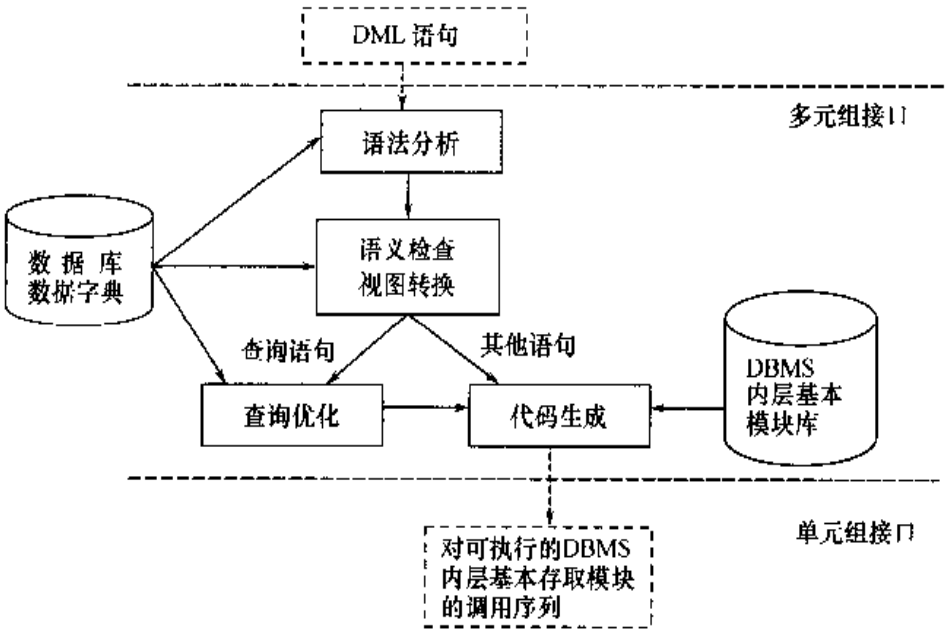


图 12.9 RDBMS 中 DML 语句的处理过程——束缚过程

首先,对 DML 语句进行词法分析和语法分析,并把外部关系名、属性名转换为内部名。外部名便于用户记忆和使用,内部名则整齐划一。在符号名转换过程中需存取数据字典。词法和语法分析通过后便生成语法分析树。

接着,根据数据字典中的内容进行查询检查,包括审核用户的存取权限和完整性检查。

对那些具有存取谓词的存取权限,它们可能与数据的值有关,则此时不能确定该语句能否执行,于是还要生成相应的动作,以便运行时检查。

完整性检查是查询检查的重要内容。RDBMS 参照数据字典中完整性约束规则,这时只是进行部分的静态约束检查,如检查数据的类型、范围是否符合数据定义。许多完整性约束条件是在执行时检查的。例如,实体完整性约束,是在执行数据插入时检查的,即检查插入的元组其主码是否已经存在,以保证主码的唯一性。同样,这时也要检查参照完整性约束(若有的话)。还有,对某些动态完整性规则,它们与数据值和执行过程有关,则也要在操作执行时进行检查。

查询检查还包括视图消解,也称为视图转换。视图转换是指,若 DML 语句涉及对视图的操纵,则首先要从数据字典中取出视图的定义,根据该定义把对视图的操作转换为对基本表的操作。

然后,对查询进行优化。优化分为两类,一类为代数优化,另一类为存取路径优化。后者要根据数据字典中记载的各种信息,按照一定的优化策略选择一个系统认为是"较好"的存取方案,并把选中的方案描述出来。

综上所述,将数据库 DML 语句转换成一串可执行的存取动作的这一过程,称为一个逐步**束缚的**(Bind)过程。它将 DML 高级的描述型语句(集合操作)转换为系统内部的低级的单元组操作,将具体的数据结构、存取路径、存储结构等结合起来,构成了一串确定的存取动作。

在各种具体的 RDBMS 中,这一束缚过程基本一致,但进行这一过程的时间则有所不同。有的在运行中进行,有的在运行前进行,于是形成了两种基本的翻译方法:解释方法和预编译方法。

12.4.2 解释方法

解释执行方法的具体做法是:直到执行前,数据库 DML 语句都以原始字符串的形式保存。当执行到该语句时,才利用解释程序去完成图 12.7 中所示的全部过程,同时予以执行。这种方法通过尽量推迟束缚过程来赢得数据独立性。

解释方法具有灵活、应变性强的优点,甚至能适应在解释过程中发生的数据结构、存储结构等的变化,因此能保持较高的数据独立性。但由于每次执行一个 DML 语句时都要执行图 12.9 中的所有步骤,尤其当这样的语句位于一个循环体内时,就要多次重复解释一个 DML 语句,开销会很大,因此效率比较低。

随着数据库技术的发展,这种方法已逐步被预编译方式所取代。

12.4.3 预编译方法

预编译方法的基本思想是:在用户提交 DML 语句之后,在运行之前对它进行翻译处理,保存产生好的可执行代码。当需要运行时,取出保存的可执行代码加以执行。下面以 DB2 为例说明这种方法。图 12.10 描述了 DB2 处理嵌入 SQL 语句的应用程序的过程。

假定被处理的对象是一个 C 语言源程序 P,它首先被一个预编译器处理。这个预编译器的功能是识别 SQL 语句,把它们替换成源语言的 CALL 语句(原来的 SQL 语句作为注解放在修改后的源程序中,以备恢复和阅读用)。预编译器还取出 SQL 语句,将每个 SQL 语句生成一个数据库请求模块 DBRM(Data Base Request Module),它是下一步束缚过程的输入。

修改后的 C 源程序按通常方式被编译、连接(Linkage Editor)处理,生成装载模块。

束缚过程类似图 12.9 执行的功能。它实际上是一个 SQL 语句的优化编译器,编译一个至多个 DBRM,生成应用规划。应用规划包含了对基本存取模块的调用,完成 DBRM 对应的源 SQL 语句的功能。

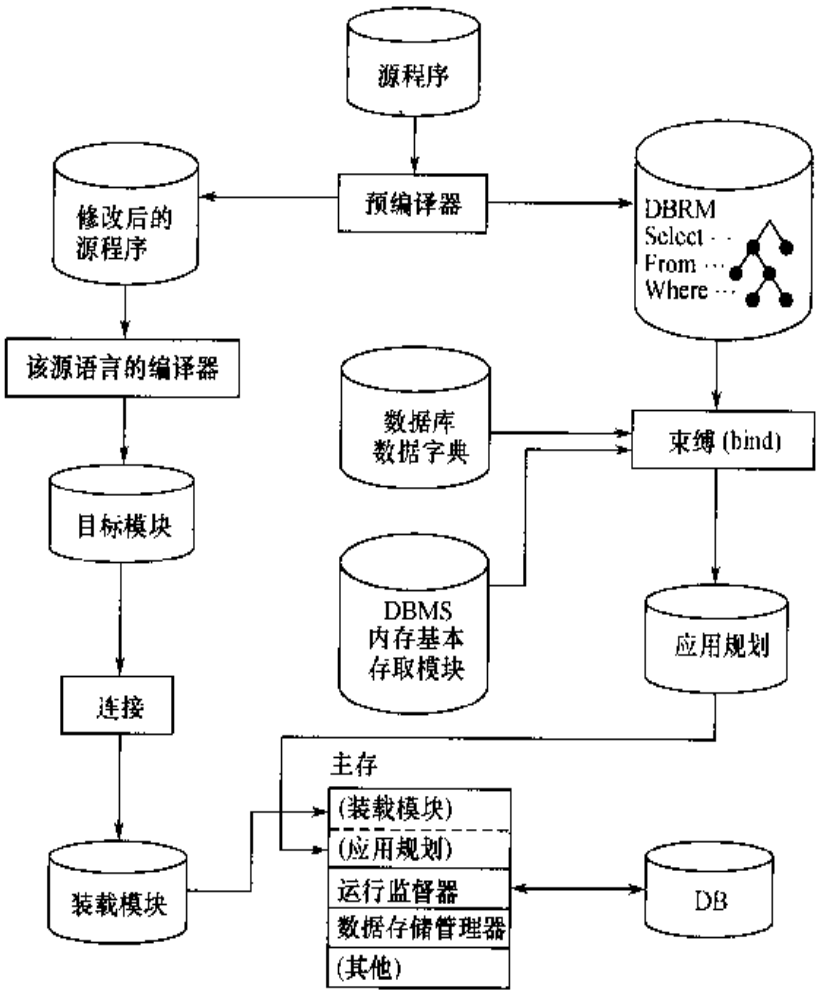


图 12.10 DB2 处理带有 SQL 语句的宿主语言源程序的过程

这样，一个 C 源程序现在分成了两部分：装载模块和应用规划。

在运行时，当装载模块执行过程中遇到数据库调用的 CALL 语句后，就把控制权交给运行监督器。运行监督器找到相应的应用规划后将其调入内存并转让控制权，然后由应用规划去启用数据存储管理器执行实际的数据存取，并将结果返回给 C 程序。

使用这种方法会遇到这样的问题：在束缚过程中进行优化所依据的条件可能在运行前已不存在，或者数据库结构已被修改，因而导致已作出的应用规划在执行时不再有效。例如，在束缚过程中决定使用某一索引来加快存取速度，而在该程序编译完成之后，运行之前，该索引被删除了，那么，运行时就会出现不可预测的现象。

为了解决这类问题，采用了重编译方法。即当数据库中因某些成分的改变而使一些程序（语句）的编译结果无效时，再对它们执行一次编译。重编译可在不同时刻进行。为了提高整个系统的效率，不应在数据库某一成分改变后就马上对受影响的那些源程序重编译，较好的方法是将受影响的编译结果置“无效”标志，在其被执行时才进行自动重编译。

自动重编译技术使得预编译方法既拥有了编译时进行束缚所带来的高效率，又具备了执行时束缚带来的数据独立性。实践证明，预编译方法的效率比其他方法高两倍以上。

12.5　数据存取层

按照 12.3.1 小节中 RDBMS 的层次结构，下面介绍数据存取层的结构以及它的各个功能子系统。

数据存取层介于语言处理层和数据存储层之间。它向上提供单元组接口，即导航式的一次一个元组的存取操作；向下则以系统缓冲区的存储器接口作为实现基础，其接口关系如图 12.11 所示。

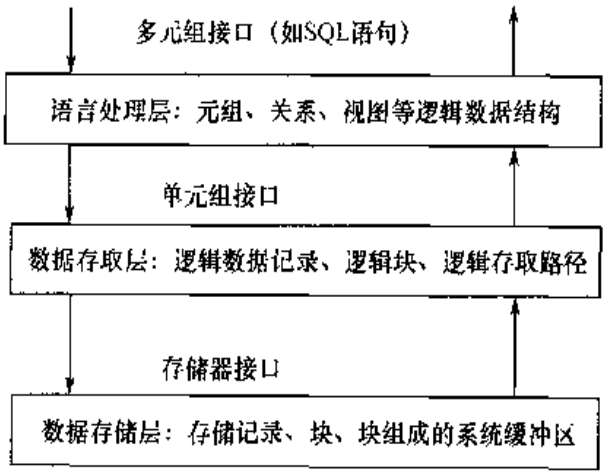


图 12.11　数据存取层和它的上下接口关系

图 12.11 中给出了每个层次中操作对象的数据结构。例如，数据存取层所涉及的主要

数据结构为逻辑数据记录、逻辑块、逻辑存取路径。

存取层的任务主要包括：

(1) 提供一次一个元组的查找、插入、删除、修改等基本操作。

(2) 提供元组查找所循的存取路径以及对存取路径的维护操作，如对索引记录的查找、插入、删除、修改。若索引是采用 B + 树，则应提供 B + 树的建立、查找、插入、删除、修改等功能。

(3) 对记录和存取路径的封锁、解锁操作。

(4) 日志文件的登记和读取操作。

(5) 其他辅助操作，如扫描、合并/排序，其操作对象有关系、有序表、索引等。

为了完成上述功能，通常把数据存取层又划分为若干功能子系统加以实现。

12.5.1 数据存取层的系统结构

数据存取层包括许多功能，在实际的 RDBMS 中由多个功能子系统来完成。图 12.12 是数据存取层的系统结构，它包括下列子系统和模块：

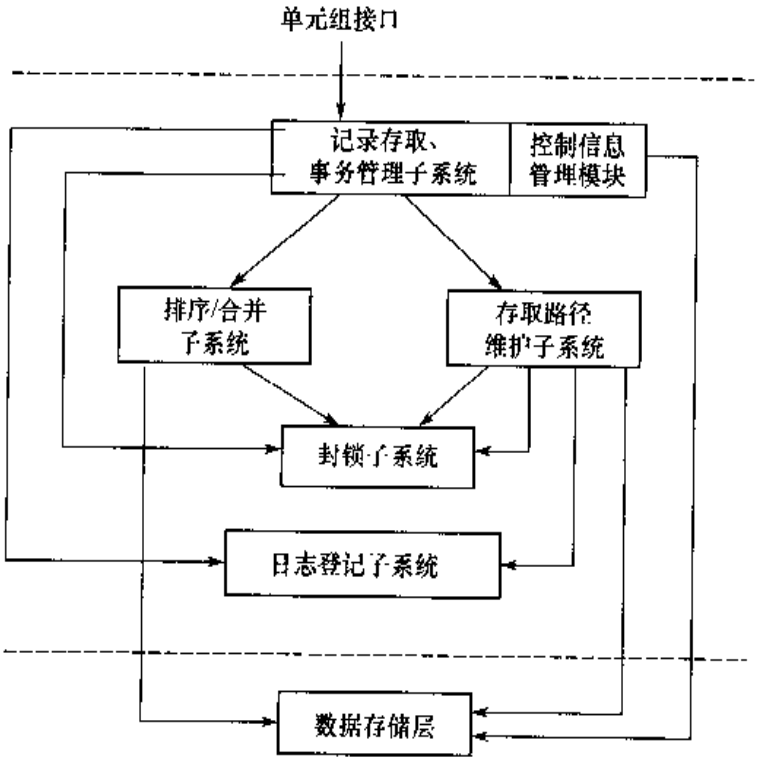


图 12.12 数据存取层的结构

- 记录存取、事务管理子系统；
- 控制信息管理模块；
- 排序/合并子系统；
- 存取路径维护子系统；

- 封锁子系统,执行并发控制;
- 日志登记子系统,用以执行恢复任务。

这些子系统相互配合,紧密联系,构成了一个完整的数据存取系统。

12.5.2 数据存取层的功能子系统

数据存取层中许多子系统的功能,如封锁子系统、日志登记子系统等已在前面章节做了介绍,下面只做扼要介绍。

一、记录存取、事务管理子系统

数据存取层不涉及存储分配、存储结构及有关参数,只在数据的逻辑结构上操作,因而可以把各种物理实现形态隐蔽起来。

记录存取子系统提供按某个属性值直接取一个元组和顺序取一个元组的存取原语。这种存取运算是按已选定的某个逻辑存取路径进行的,如某个数据文件或某个索引。这类存取操作的例子有:

- 在某个存取路径上按属性值找元组(FIND);
- 按相对位置找元组(NEXT,PRIOR,FIRST,LAST);
- 给某关系增加一个元组(INSERT);
- 从找到的元组中取某个属性值(GET);
- 从某关系中删去一个元组(DELETE);
- 把某修改完的元组写回关系中(REPLACE)。

事务管理子系统提供定义和控制事务的操作。数据库中事务是并发控制和恢复的单位。事务管理的基本操作有:

- 定义事务开始(BEGIN TRANSACTION);
- 事务提交(COMMIT);
- 事务滚回(ROLLBACK)。

事务管理子系统提供的这些操作将登记进日志文件中。

二、日志登记子系统

日志登记子系统和事务管理子系统紧密配合,完成 RDBMS 对事务和数据库的恢复任务,它把事务开始、滚回、提交;对元组的插入、删除、修改;对索引记录的插入、删除、修改等每一个操作作为一个日志记录存入日志文件中。当事务或系统软、硬件发生故障时利用日志文件执行恢复。与日志文件有关的主要操作有:

- 写日志记录(WRITELOG);
- 读日志记录(READLOG);
- 扫描日志文件(SCANLOG);
- 撤销尚未结束的事务(UNDO);

• 重做已经结束的事务(REDO)。

三、控制信息管理模块

该模块利用专门的数据区(内存中)登记不同记录类型以及不同存取路径的说明信息(取自数据字典)和控制信息。这些信息是存取元组和管理事务的依据。它和事务管理、记录存取子系统一起保证事务的正常运行。

该模块提供对数据字典中说明信息的读取、增加、删除和修改操作。

四、排序/合并子系统

在语言处理层中,描述性语言表达的集合级操作被转换成一系列的对数据存取层所提供的存取原语的调用。为了得到用户所要求的有序输出,为了加速关系运算(如自然连接)的中间步骤,常常需要对关系元组重新排序。这一工作由排序/合并子系统来完成。下面列举排序操作的若干主要用途。

1. 输出有序结果

例如,用户提出如下查询要求:

```
SELECT Eno,Salary
FROM EMP
ORDER BY Salary DESC;
```

若 EMP 表上的 Salary 属性已建有索引(如 B + 树),则可以顺序扫描索引获得要求的输出。若 Salary 上没有索引,则必须对 EMP 表按 Salary 的属性值重新降序排序,以得到所要的结果。

2. 删去重复值

删去重复值(简称删重)是并、交、差、属于、不属于等关系运算中常常首先要做的预处理。对无序数据,删重的代价一般为 $O(n^2)$ 数量级,比较昂贵。因此,用先排序后删重或边排序边删重的方法可以降低开销。

3. 支持排序 - 合并方法的连接操作

对两个无序关系的连接运算有时可以用排序 - 合并方法来提高连接运算速度,即先对两个关系按连接属性值排序,然后对两个有序关系执行连接。

4. 支持动态建立索引结构

数据库中常用的索引结构有 B + 树。B + 树的叶页的索引记录形式为(码值,TID),其中 TID 为元组标识符。TID 可用元组逻辑记录号或主码值或数据块号加位移等来表示。索引记录在 B + 树的叶页上是顺序存储的,因此在初建 B + 树索引时首先要对(码值,TID)排序。

5. 减少数据块的存取次数

通过 B + 树索引存取元组时,首先得到(码值,TID)集合,然后根据 TID 存取相应的元组。当 TID 是用数据块号加位移来表示时,可以首先对 TID 排序,使相同或临近块号的 TID 聚集在一起,然后按数据块号顺序存取物理数据块,避免无序状态下重复读块的情况,减少

数据块的存取次数。

排序操作的用途还很多,这里不一一列举了。

由此可见,排序操作是数据存取子系统和存取路径维护子系统都要经常调用的操作。它对提高系统效率具有关键的作用。因此,排序子系统的设计十分重要,应采用高效的外排序算法。因为排序的数据量很大,要使用外排序算法。

五、存取路径维护子系统

对数据执行插入、删除、修改操作的同时要对相应的存取路径进行维护。例如,若用B+树索引作为存取路径,则对元组进行插入、删除、修改操作时要对该表上已建立的所有B+树索引进行动态维护,插入、删除相应的索引项。否则,就会造成B+树索引与数据库表的不一致,当再通过B+树索引结构存取元组时便会造成操作失败或错误结果。

六、封锁子系统

封锁子系统完成并发控制功能。有关封锁的概念和技术,包括封锁的类型、封锁相容矩阵、死锁处理、可串行性准则、两段锁协议等已在前面章节详细讨论,这里只说明两点:

第一,在操作系统中也有并发控制问题,其实现并发控制的方法通常也采用封锁技术。DBMS封锁技术和操作系统的封锁技术相比(表12.1),内容更加丰富,技术更加复杂。

表12.1 操作系统和数据库管理系统封锁技术的比较

	操作系统	数据库管理系统
封锁对象	单一,系统资源(包括CPU、设备、表格等)	多样,数据库中各种数据对象(包括用户数据、索引(存取路径)、数据字典等)
封锁对象的状态	静态、确定、各种封锁对象在封锁表中占有一项。封锁对象数是不变的	动态,不确定。封锁对象动态改变着、常常在执行前不能确定。一个封锁对象只有当封锁时才在封锁表中占据一项
封锁的粒度	不变,由于封锁对象单一、固定,封锁粒度不会改变	可变,封锁可加到或大或小的数据单位上,封锁粒度可以是整个数据库、记录或字段
封锁的类型	单一,排它锁	多样,一般有共享锁(S Lock)、排它锁(X Lock)或其他类型的封锁,随系统而异

第二,DBMS中封锁子系统设计的难点不仅在于技术复杂,而且在于其实现手段依赖于操作系统提供的环境,例如封锁表的设计。由于封锁表必须能为多个进程共享,能动态建立和释放,因此封锁表的设计就随操作系统环境而异。它是封锁子系统设计的关键。

12.6 缓冲区管理

数据存取层的下面是数据存储区(简称存储层)。存储层的主要功能是存储管理,包括缓冲区管理、内外存交换、外存管理等。其中缓冲区管理是十分重要的。存储层向存取层提

供的接口是由定长页面组成的系统缓冲区。

系统缓冲区的设立是出于两方面的原因：一是它把存储层以上各系统成分和实在的外存设备隔离。外存设备的变更不会影响其他系统成分，使 RDBMS 具有设备独立性。二是提高存取效率。RDBMS 利用系统缓冲区滞留数据。当存取层需要读取数据时存储子系统首先到系统缓冲区中查找。只有当缓冲区中不存在该数据时才真正从外存读入该数据所在的页面。当存取层写回一元组到数据库中时，存储子系统并不把它立即写回外存，仅把该元组所在的缓冲区页面作一标志，表示可以释放。只有当该用户事务结束或缓冲区已满需要调入新页时才按一定的淘汰策略把缓冲区中已有释放标志的页面写回外存。这样可以减少内外存交换的次数，提高存取效率。

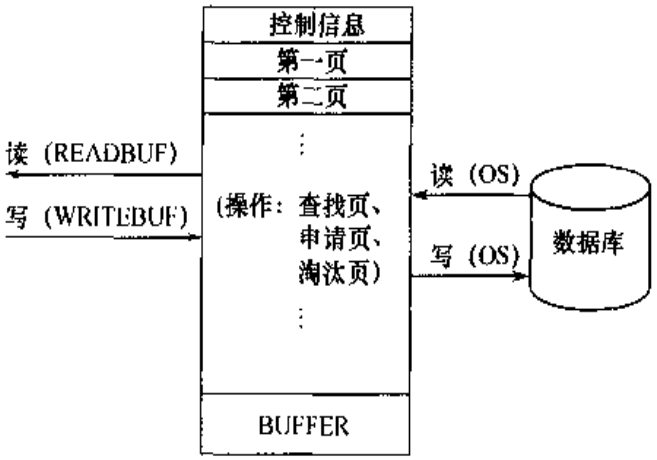


图 12.13 数据库缓冲区及上下接口

系统缓冲区可由内存或虚存组成。由于内存空间紧张，缓冲区的大小、缓冲区内存和虚存部分的比例要精心设计。针对不同的应用和环境按一定的模型进行调整。既不能让缓冲区占据太大内存空间，也不能因其空间太小而频频缺页、调页，造成“抖动”，影响效率。图 12.13 给出了缓冲区及上下接口示意图。

缓冲区由控制信息和若干定长页面组成。缓冲区管理模块向上层提供的操作是缓冲区的读(READBUF)、写(WRITEBUF)。缓冲区内部的管理操作有：查找页、申请页、淘汰页。缓冲区管理调用操作系统的操作有：读(READ)、写(WRITE)。以读操作为例，缓冲区管理的大致过程如图 12.14 所示。可以看到缓冲区管理中主要算法是淘汰算法和查找算法。操作系统中许多淘汰算法可以借鉴，如 FIFO(先进先出算法)、LRU(最近最少使用的先淘汰算法)以及它们的各种改进算法。查找算法用来确定所请求的页是否在内存，可采用顺序扫描、折半查找、

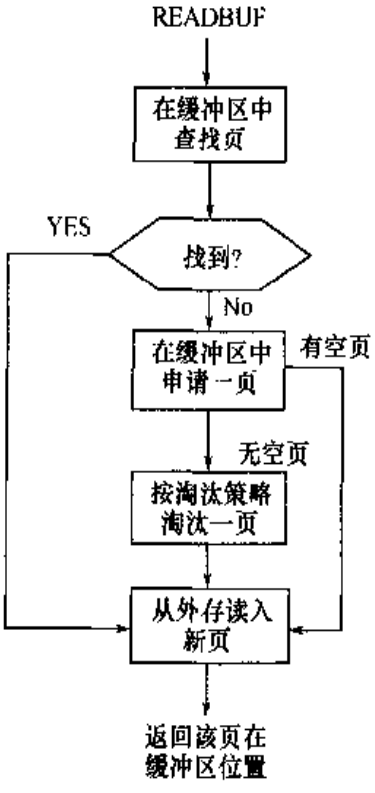


图 12.14 缓冲区管理示意图(读操作)

Hash 查找算法等。

12.7 数据库物理组织

从 DBMS 最外层语言处理层开始逐步深入地讨论了数据存取层、数据存储层各层的系统结构、功能、任务以及实现技术。现在进入数据库本身，介绍数据库的物理组织。

数据库是大量数据的有结构的综合性集合。如何将这样一个庞大的数据集合以最优的形式组织起来存放在外存上是一个非常重要的问题。所谓“优”应包括两方面：一是存储效率高，节省存储空间；二是存取效率高，速度快，代价小。

数据库实现的基础是文件，对数据库的任何操作最终要转化为对文件的操作。所以在数据库物理组织中，基本的问题是如何设计文件组织或者利用操作系统提供的基本的文件组织方法。

这里不打算讨论文件组织的基本方法和常用的文件组织形式，这些内容在数据结构课程中已介绍。这一节主要讨论如何利用基本的文件组织方法来实现数据库组织。

数据库系统是文件系统的发展。文件系统中每个文件存储同质实体的数据，各文件是孤立的，没有体现实体之间的联系。数据库系统中数据的物理组织必须体现实体之间的联系，支持数据库的逻辑结构——各种数据模型。因此数据库中要存储 4 个方面的数据：

- 数据描述，即数据外模式、模式、内模式；
- 数据本身；
- 数据之间的联系；
- 存取路径。

这 4 个方面的数据内容都要采用一定的文件组织方式组织、存储起来。

一、数据字典(DD)的组织

有关数据的描述存储在数据库的数据字典中。数据字典的特点是数据量比较小(与数据本身比)、使用频繁，因为任何数据库操作都要参照数据字典的内容。数据字典在网状、层次数据库中常常用一个特殊的文件来组织。所有关于数据的描述信息存放在一个文件中，例如 HP3000 计算机系统中 IMAGE 网状数据库的模式就是用一个称为“根文件”的特权文件来存放的。关系数据库中数据字典的组织通常与数据本身的组织相同。数据字典按不同的内容在逻辑上组织为若干张表(如图 12.8 所示)，在物理上就对应若干文件而不是一个文件。由于每个文件中存放数据量不大，可简单地用顺序文件来组织。

二、数据及数据联系的组织

关于数据自身的组织，DBMS 可以根据数据和处理的要求自己设计文件结构，也可以从操作系统提供的文件结构中选择合适的加以实现。目前，操作系统提供的常用文件结构有：顺序文件、索引文件、索引顺序文件、Hash 文件(杂凑文件)和 B 树类文件等。

数据库中数据组织与数据之间联系是紧密结合的。在数据的组织和存储中必须直接或间接、显示或隐含地体现数据之间的联系，这是数据库物理组织中主要考虑和设计的内容。

在网状和层次数据库中，常用邻接法和链接法实现数据之间的联系。对应到物理组织方式中，就要在操作系统已有的文件结构上实现数据库的存储组织和存取方法。例如，IMS数据库中，操作系统提供的低级的存取方法有 SAM、ISAM、VSAM、OSAM。IMS 数据库管理系统在此基础上设计了 HSAM、HISAM、HDAM、HIDAM 四种数据库的存储组织和相应的存取方法。其中 HSAM 层次顺序存取方法按照片段值的层次序列码的次序顺序存放各片段值。而层次序列码正体现了数据之间的父子和兄弟联系。这是一种典型的按物理邻接方式实现数据之间联系的方法。在这种存储方法中，整个数据库中不同片段型的数据均存储在一个 SAM 文件中。

网状数据库中最常用的组织策略是：各记录型分别用某种文件结构组织，记录型之间的联系——SET 用指引元方式实现。即在每个记录型中，增加 DBMS 控制和维护的系统数据项——指引元，它和用户数据项并存于同一个记录中。

关系数据库中实现了数据表示的单一性。实体及实体之间的联系都用一种数据结构——“表”来表示。在数据库的物理组织中，每一个表通常对应一种文件结构。因此数据和数据之间的联系两者组织方式相同。

三、存取路径的组织

在网状和层次数据库中，存取路径是用数据之间的联系来表示的，因此已与数据结合并固定下来。

关系数据库中，存取路径和数据是分离的，对用户是隐蔽的。存取路径可以动态建立、删除。存取路径的物理组织通常采用 B 树类文件结构和 Hash 文件结构。在一个关系上可以建立若干个索引。有的系统支持组合属性索引，即在两个或两个以上属性上建立索引。索引可以由用户用 CREATE INDEX 语句建立，用 DROP INDEX 删除。在执行查询时 DBMS 查询优化模块也会根据优化策略自动地建立索引，以提高查询效率。由此可见，关系数据库中存取路径的建立是十分灵活的。

12.8 小 结

本章主要讨论 DBMS 的基本功能、系统结构及主要的实现技术，是 DBA 和数据库应用系统开发人员应该掌握的内容。

按照 RDBMS 的层次结构依次介绍了语言处理层、数据存取层、缓冲区管理层、数据物理组织层等各个层次的主要任务和功能、涉及的主要概念和问题。这里没有讲解 DBMS 具体的实现技术、实现算法和数据结构，这些内容应该在专门的一本 DBMS 原理和设计中讲解。

数据库领域过去所取得的主要成就之一，就是数据建模和 DBMS 核心的研究与开发。

40 多年来数据库研究人员深入研究了这些技术并且取得了辉煌的成就。例如，人们已经清楚地知道如何在外部存储设备上存储数据、如何分片；如何使用各种复杂的存取方法、缓冲策略和索引技术访问外部存储设备上的数据；数据库恢复、并发控制、完整性和安全性的实施、查询处理和优化等技术也得到深入了解，并在大多数商用的集中式和分布式 RDBMS 中得以实现。

从这一章看到了在数据库领域从理论的创立和工作原理的推出，到实验研究和原型系统的研制，最后演化为商品的成功过程。数据库系统研究的历史有力地说明进行基础研究可以带来商业上的成功。

习 题

1. 试述 DBMS 的基本功能。
2. 试述 DBMS 四种进程组织方案，并分析各种方案的特点和优缺点。
3. 理解并解释下列术语的含义：进程、任务、“轻权”进程、线程、线索。
4. 什么是 DBMS 的多线索机制？有什么优点？
5. RDBMS 工作过程是什么？给出 DBMS 插入一个记录的活动过程，画出活动过程示意图。
6. RDBMS 的语言处理层是如何处理一个 DDL 语句的？
7. 试述 RDBMS 的语言处理层处理一个 DML 语句的大致过程。
8. 什么是处理 DML 语句的解释方法和预编译方法？试述二者的区别、联系，比较各自的优缺点。
9. 试述数据存取层主要的子系统及其功能。
10. 在操作系统中也有并发控制问题，为什么 DBMS 还要并发控制机制？
11. 试比较 DBMS 与操作系统的封锁技术。
12. DBMS 中为什么要设置系统缓冲区？
13. 数据库中要存储和管理的数据内容包括哪些方面？

*14. 请给出缓冲区管理中的一个淘汰算法，并上机实现（提示：你首先要设计缓冲区的数据结构，然后写出算法）。

*15. 请写出对一个文件按某一个属性的排序算法（设该文件的记录是定长的），并上机实现。若要按多个属性排序，你能写出改进的算法吗？

*16. 请给出 B + 树文件的创建和维护（增、删、改）算法并上机实现（提示：设 B + 树叶结点上仅存放索引项[码值，TID]，你首先要设计索引项，B 树叶页和非叶页的数据结构，然后写出算法）。注：习题 15、16、17 可作为学生 DBMS 课程的实习课题。这些模块是 DBMS 中必不可少的基本模块。

本章参考文献

1 Gray Jamed. Notes on Data Base Operation Systems. In Operating Systems: An Advanced Course. Springer Verlag, 1978

（本文献系统地综述了数据库系统及其与操作系统有关的问题。）

2 Stonebraker M R. Operating System Support for Database Management. CACM VOL. 24, No. 7, 1981

（本文提出了操作系统应提供对数据库系统更好支持的问题。）

3 Traiger I L. Virtual Memory for Database Systems. ACM SIGOPS Operating Systems Review, Volume 16, Number 4, 1982

（本文讨论了数据库系统中与操作系统有关的问题。）

4 马文赛. DBMS 进程结构研究及多线索 DBMS 的设计与实现. 中国人民大学硕士论文, 1993

（本文详细讨论了数据库管理系统的进程结构和多线索 DBMS 的概念、优点和实现技术。）

5 Astrahan M M et al. System R: Relational Approach to Database Management. ACM TODS 1, No. 2, 1976

（本文详细介绍了 IBM 公司的 System R 的实现技术。）

6 Blasgen M M et al. System R: An Architectural Overview. IBM Sys. J. 20, No. 1, 1981

（本文详细介绍了 IBM 公司的 System R 的系统结构。）

7 Selinger P G et al. Access Path Selection in a Relational Database Management System. Proc. of ACM SIGMOD International Conference on Management of Data, 1979

8 Blasgen M M. Eswaran K P. Storage and Access in Relational Database. IBM Sys. J. 16, No. 4, 1977

9 Merrett T H. Why Sort/Merger Gives the Best Implementation of the Natural Join. ACM SIGMOD Record 13, No. 2, 1983

（本文阐述了自然连接的实现技术之一，排序 - 合并方法及其优点。）

10 Bayer R, McCreight C. Organization and Maintenance of Large Ordered Indexes. Acta informatica 1, No. 3, 1972

（Bayer 和 McCreight 在本文中首先提出了 B 树存储结构，现在 B 树已广泛用于数据库的物理组织之中。）

11 Comer D. The Ubiquitous B-tree. ACM Computing Surveys 11, No. 2, 1979

（本文对 B 树的历史和发展进行了系统的综述，并介绍了各种 B 树的变种。）

12 王珊，吴鸥琦. B 树及其在文件组织中的应用. 电子计算机动态, 1981

（本文讨论 B 树的优缺点及其作用。）

13 王珊，吴鸥琦. B + 树效率分析和组织聚集索引的算法——B + 树的进一步研究和应用. 计算机研究与发展, 1982

（本文讨论了 B + 树的组织和算法。）

14 Wagner R E. Indexing Design Considerations. IBM sys. J. 12, No. 4, 1973

15 Knott G D. Expandable Open Addressing Hash Table Storage and Retrieval. Proc. of ACM SIGFIDET Workshop on Data Description, Access and Control, 1971

16 Fagin R, Nievergelt J, Pippenger N et al. Extendible Hashing: A Fast Access Method for Dynamic Files. ACM TODS 4, No. 3, 1979

17 Lum V. Multi-Attribute Retrieval with Combined Indexes. CACM 13, No. 11, 1971

（本文研究了在多属性上建立组合属性索引的问题。）

18 Knuth D E. The Art of Computer Programming. Vol. Ⅲ, 1973

（本书是一套著名丛书中的一本，在本书中 Knuth 对多种数据存取方法进行了详细的分析，包括对 B + 树及其相关存储结构的分析。）

19 Wirth W. Programming = Algorithm + Data Structures, 1976

（本书是一本数据结构教科书，其中详细给出了 B 树的增加、删除、修改算法。）

20 王珊. 关系数据库产品透析. 中国计算机报专家评述, 1993

（本文分析并给出了关系数据库产品的 3 个发展阶段，总结了各个阶段的特点。）

21 孟小峰，王珊. 嵌套查询的非嵌套化处理研究. 计算机学报. Vol. 18, No. 4, 1995

（本文研究了 JA 类嵌套查询的优化问题，提出了把嵌套查询非嵌套化的算法。）

22 项明峰，王珊. 数据库字典的封锁技术. 第 10 届全国数据库会议论文集, 1992

（本文提出了一种有效的数据库字典的封锁方法，并成功地用于 COBASE 数据库管理系统的实现之中。COBASE 是北京大学、中国人民大学、中国软件总公司等单位“八五”攻关项目的科技成果。）

第四篇 新技术篇

本篇概述数据库的发展历程，介绍数据库的新技术和新进展，从宏观概述到具体若干新技术、新系统进行讲解，使读者在学习前面3篇基本知识和技术的基础上开阔思路和眼界，为进一步学习和研究做必要的准备和铺垫。

新技术篇包括5章。

第十三章是数据库技术发展的总体轮廓，首先概述数据库发展的光辉历程，其次阐明数据库技术与其他技术相互渗透与融合以及数据库应用领域不断拓宽又日益深入的鲜明特点，最后给出当前数据库技术发展趋势和若干研究热点。

接下来的4章选择了当前较重要的新技术和新系统进行概要的介绍，它们是第十四章分布数据库系统、第十五章对象关系数据库系统、第十六章XML数据库，以及第十七章数据仓库和联机分析技术。其中对象关系数据库系统、XML数据库和联机分析技术是SQL标准新增加的内容。

第十三章 数据库技术新发展

本章第一部分以数据模型、数据库应用、数据库管理系统开发技术3个方面为主线概述数据库的发展历程，展示数据库学科在理论、应用、系统开发等研究和应用领域的主要内容与发展方向。其目的在于提供一个宏观的、总体的数据库学科的视图，使读者既能了解数据库的新进展又能了解数据库技术的发展过程，既能了解新的数据库分支的基本内容，又能了解这些分支之间的相互联系。

本章第二部分简单给出当前数据库技术的发展趋势和若干热门的研究课题。

数据库技术从“理论研究”到“原型开发与技术攻关”再到“实际产品研制和应用”，已经形成良性循环，成为计算机领域的成功典范。吸引了学术界和工业界众多的科技人员，使得数据库研究日新月异，新技术、新系统层出不穷，科技队伍也不断壮大。

13.1 数据库技术发展概述

数据库技术产生于20世纪60年代中期，到今天为止仅仅40多年的历史，却已经历了3代演变，造就了C. W. Bachman、E. F. Codd和James Gray三位图灵奖得主，发展了以数据建模和DBMS核心技术为主，内容丰富的一门学科，带动了一个巨大的软件产业——DBMS产品及其相关工具和解决方案。这40多年可谓成就辉煌。

数据库技术是计算机科学技术中发展最快的领域之一，也是应用最广的技术之一，目前，它已成为计算机信息系统与应用系统的核心技术和重要基础。

当今数据库系统是一个大家族，数据模型丰富多样，新技术内容层出不穷，应用领域日益广泛，当读者步入数据库领域时，面对众多复杂的数据库系统难免产生迷惑和混乱。

图13.1通过一个三维空间的视图从数据模型、新技术内容、应用领域3个方面，描述了数据库系统的发展、特点和相互关系。

数据模型是数据库系统的核心和基础。按照数据模型的进展，数据库技术可以相应地分为3个发展阶段，即第一代的网状、层次数据库系统，第二代的关系数据库系统，以及发展到今天以第三代数据库系统为核心的数据库大家族。

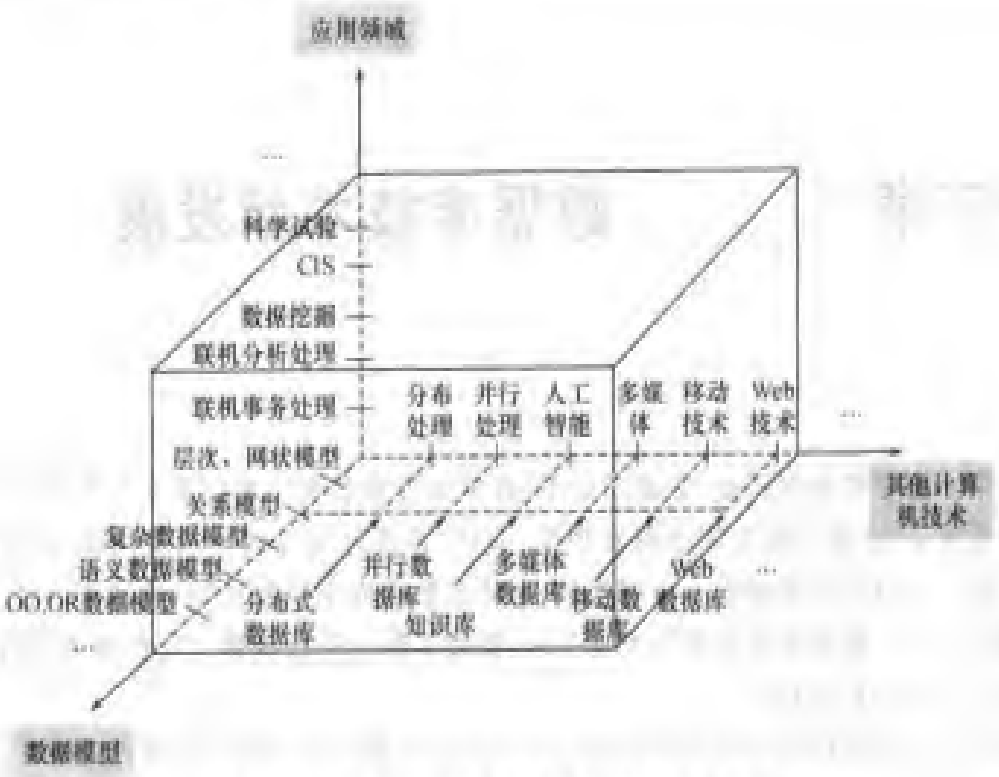


图 13.1 数据库系统的发展和相互关系示意图

13.2 数据库发展的 3 个阶段

虽然数据库系统模型分为层次模型和网状模型，但实质上层次模型是网状模型的特例，它们都是格式化模型。它们从体系结构、数据库语言到数据存储管理均具有共同特征，是第一代数据库系统。

13.2.1 第一代数据库系统

第一代数据库系统指层次和网状数据库系统，其代表系统是：

1. 1969 年由 IBM 公司研制的层次模型的数据库管理系统 IMS(Information Management System)。

2. 美国数据库系统语言协商会 CODASYL(Conference On Data System Language)下属的数据库任务组 DBTG(Data Base Task Group)对数据库方法进行了系统的研究和探讨，于 20 世纪 60 年代末 70 年代初提出了若干报告，称为 DBTG 报告。DBTG 报告确定并建立了数据库系统的许多概念、方法和技术。DBTG 所提议的方法是基于网状结构的，它是数据库网状模型的典型代表。

可以说，层次数据库是数据库系统的先驱，而网状数据库则是数据库概念、方法、技术的

奠基者。它们是数据库技术中研究得最早的两种数据库系统。两者的区分是以数据模型为基础,层次数据库的数据模型是分层结构的,而网状数据库的数据模型是网状的,它们的数据结构都可以用图来表示。层次数据模型对应于有根定向有序树,而网状模型对应的是有向图。所以,这两种数据模型可以统称为格式化数据模型。

这两种数据库系统具有以下几个共同特点。

1. 支持三级模式的体系结构

三级模式通常指外模式、模式、内模式。模式之间具有转换(或称为映射)功能。

2. 用存取路径来表示数据之间的联系

这是数据库系统和文件系统的主要区别之一。

数据库不仅存储数据,而且存储数据之间的联系。数据之间的联系在层次和网状数据库系统中都是用存取路径来表示和实现的。

3. 独立的数据定义语言

层次数据库系统和网状数据库系统有独立的数据定义语言,用以描述数据库的外模式、模式、内模式以及相互映像。诸模式一经定义,就很难修改。

4. 导航的数据操纵语言

层次和网状数据库的数据查询和数据操纵语言是一次一个记录的导航式的过程化语言。这类语言通常嵌入某一种高级语言,如 COBOL、FORTRAN、PL/1、C 语言中。

所谓导航,就是指用户不仅要了解"干什么",而且要指出"怎么干"。用户必须使用某种高级语言编写程序,一步一步地"引导"程序按照数据库中某一条预先定义的存取路径来访问数据库,最终达到要访问的数据目标。在访问数据库时,每次只能存取一条记录值。

导航式的数据操纵语言,优点是存取效率高,缺点是编程繁琐,给用户编写应用程序带来了困难。其设计在很大程度上依赖于设计者自己的经验和实践,因而只有具有计算机专业水平的应用程序员才能掌握和使用这类数据库操纵语言。此外,应用程序的可移植性较差,数据的逻辑独立性也较差。

13.2.2 第二代数据库系统

支持关系数据模型的关系数据库系统是第二代数据库系统。

1970 年,IBM 公司 San Jose 研究室的研究员 E. F. Codd 发表了题为《大型共享数据库数据的关系模型》论文,提出了数据库的关系模型,开创了数据库关系方法和关系数据理论的研究,为关系数据库技术奠定了理论基础。

20 世纪 70 年代是关系数据库理论研究和原型开发的时代。其中以 IBM San Jose 研究室开发的 System R 和 Berkeley 大学研制的 INGRES 为典型代表。这一年代,人们经过大量的高层次的研究和开发取得了以下几个方面的主要成果。

1. 奠定了关系模型的理论基础,给出了人们一致接受的关系模型的规范说明。

2. 研究了关系数据语言，包括关系代数、关系演算、SQL 及 QBE 等。这些描述性语言一改以往程序设计语言和网状、层次数据库系统中数据库语言的风格，以其易学易懂的优点得到了最终用户的喜爱，为 20 世纪 80 年代数据库语言的标准化打下了基础。

3. 研制了大量的 RDBMS 的原型，攻克了系统实现中查询优化、并发控制、故障恢复等一系列关键技术。这不仅大大丰富了 DBMS 实现技术和数据库理论，更促进了 RDBMS 产品的蓬勃发展与广泛应用。

在 20 世纪 70 年代，关系数据库系统从实验室走向了社会，因此，在计算机领域中有人把这一年代称为数据库时代。在接下来的 20 世纪 80 年代，几乎所有新开发的 DBMS 均是关系数据库。

关系数据库是以关系模型为基础的。关系模型概括地讲由以下三部分组成。

1. 数据结构

数据结构包括域及域上定义的关系等。关系模型中，实体、实体与实体之间的联系都通过关系这种单一的结构类型来表示。

2. 关系操作

关系操作的能力可用关系代数(或等价的关系演算)中并、交、差、广义笛卡尔积、选择、投影、连接、除等操作来表示。关系操作的特点是集合操作，无论是操作的对象还是操作的结果都是集合。这种操作方式被称为一次一集合(set-at-a-time)的方式，与非关系型的一次一记录(record-at-a-time)的方式相对照。

3. 数据完整性

数据完整性包括实体完整性、参照完整性以及与应用有关的完整性。

实体完整性和参照完整性是关系模型必须满足的完整性约束条件，应由关系数据库系统自动支持；与应用有关的完整性由用户定义，它是针对某一具体数据库的约束条件，由应用环境决定，关系数据库系统应提供定义和检验这类完整性的机制。

关系数据库系统支持关系模型。关系模型不仅简单、清晰，而且有关系代数作为语言模型，有关系数据理论作为理论基础。因此，关系数据库系统具有形式化基础好、数据独立性强、数据库语言非过程化等特色，标志着数据库技术已经发展到了第二代。

13.2.3 新一代数据库系统

第二代数据库系统的数据模型虽然描述了现实世界数据的结构和一些重要的相互联系，但是仍不能捕捉和表达数据对象所具有的丰富而重要的语义，因此尚只能属于语法模型。

第三代的数据库系统将以更丰富的数据模型和更强大的数据管理功能为特征，从而满足更加广泛复杂的新应用的要求。

新一代数据库技术的研究和发展导致了众多不同于第一、二代数据库的系统诞生，构成

了当今数据库系统的大家族。

这些新的数据库系统无论是基于扩展关系数据模型的(对象关系数据库),还是 OO 模型的,是分布式、客户/服务器还是混合式体系结构的,是在 SMP 还是在 MPP 并行机上运行的并行数据库系统,乃至是用于某一领域(如工程、统计、GIS)的工程数据库、统计数据库、空间数据库等,都可以广泛地称之为新一代数据库系统。

尽管许多学者认为第三代数据库系统尚未成熟,但这并不妨碍人们来讨论和研究什么样的数据库系统可称之为第三代数据库系统。

1990 年,高级 DBMS 功能委员会[①]发表了《第三代数据库系统宣言》的文章(以下简称《宣言》),提出了第三代 DBMS 应具有的 3 个基本特征 (《宣言》中称为 3 条基本原则)。《宣言》从 3 个基本特征导出了 13 个具体的特征和功能(《宣言》称为 13 个命题)。

1. 第三代数据库系统应支持数据管理、对象管理和知识管理

除提供传统的数据管理服务外,第三代数据库系统将支持更加丰富的对象结构和规则,应该集数据管理、对象管理和知识管理为一体。

第三代数据库系统不像第二代关系数据库那样有一个统一的关系模型。但是,有一点应该是统一的,即无论该数据库系统支持何种复杂的、非传统的数据模型,它应该具有 OO 模型的基本特征。数据模型是划分数据库发展阶段的基本依据,因此第三代数据库系统应该是以支持面向对象数据模型为主要特征的数据库系统。但是,只支持 OO 模型的系统不能称为第三代数据库系统,第三代数据库系统还应具备其他特征。

2. 第三代数据库系统必须保持或继承第二代数据库系统的技术

这就是说,必须保持第二代数据库系统的非过程化数据存取方式和数据独立性,第三代数据库系统应继承第二代数据库系统已有的技术。这不仅能很好的支持对象管理和规则管理,而且能更好地支持原有的数据管理,支持多数用户需要的即席查询等。

3. 第三代数据库系统必须对其他系统开放

数据库系统的开放性表现在:支持数据库语言标准,在网络上支持标准网络协议,系统具有良好的可移植性、可连接性、可扩展性和可互操作性等。

13.3 数据库系统发展的特点

本节按照图 13.1,从数据模型、新技术内容、应用领域三个方面来描述数据库系统发展的特点及其相互关系。

① 该委员会成员包括:加州 Berkeley 分校的 Michael Stonebraker 和 Lawrence A. Rowe,IBM 研究所的 Bruce Lindsay,Tandem 计算机公司的 James Gray,Wisconsin 大学的 Michael Carey,GTE 实验室的 Michael Brodie,DEC 公司的 Philip Bernstaein,Oracle 公司的 David Beech。

13.3.1　数据模型的发展

数据库的发展集中表现在数据模型的发展。从最初的层次、网状数据模型发展到关系数据模型，数据库技术产生了巨大的飞跃。关系模型的提出，是数据库发展史上具有划时代意义的重大事件。关系理论研究和关系数据库管理系统研制的巨大成功进一步促进了关系数据库的发展，使关系数据模型成为具有统治地位的数据模型。20 世纪 80 年代之后，几乎所有的数据库系统都是关系的，它的应用遍布各个领域。

随着数据库应用领域的扩展，数据对象的多样化，传统的关系数据模型开始暴露出许多弱点，如对复杂对象的表示能力较差，语义表达能力较弱，缺乏灵活丰富的建模能力，对文本、时间、空间、声音、图像和视频等数据类型的处理能力差等。为此，人们提出并发展了许多新的数据模型。这些尝试是沿着如下几个方向进行的。

1. 对传统的关系模型(1NF)进行扩充，引入了少数构造器，使它能表达比较复杂的数据类型，增强其结构建模能力，这样的数据模型称为复杂数据模型。按照它们进行扩充的侧重点，复杂数据模型可分为两种。

一种是偏重于结构的扩充。首先出现的这类模型是嵌套关系模型(NF^2)。它能表达“表中表”，并且表中的一个域可以是一个函数(称为虚域)。

另一种是侧重于语义的扩充，像 U. C. Berkeley 大学的 POSTGRES 系统。它支持关系之间的继承，也支持在关系上定义函数和运算符，但关系的结构仍然是一张平面表。“表中表”只能通过关系上定义的函数来模拟。

2. 新提出和发展的数据模型相比关系模型来说增加了全新的数据构造器和数据处理原语，以表达复杂的结构和丰富的语义。这类模型中比较有代表性的是函数数据模型(FDM)、语义数据模型(SDM)、RM/T 模型以及 E-R 模型等，常常统称它们为语义数据模型。它们的特点是引入了丰富的语义关联(如 ISA、ISP)，能更自然、恰当地表达客观世界中实体间的联系。此外，由于拥有比较丰富的结构构造器(如 TUPLE、LIST、SET 等)，它们也具有了很强的结构表达能力。

也许是由于语义数据模型比较复杂，在程序设计语言和技术方面没有相应的支持，因此，它们都没有在数据库系统实现方面有重大的突破，只是作为数据库设计中概念建模的一种工具(如 E-R 模型)。

3. 将上述语义数据模型和 OO 程序设计方法结合起来，我们提出了面向对象的数据模型。面向对象的数据模型吸收了面向对象程序设计方法学的核心概念和基本思想。一个面向对象数据模型是用面向对象观点来描述现实世界实体(对象)的逻辑组织、对象间限制、联系等的模型。一系列面向对象核心概念构成了面向对象数据模型的基础。

对象关系数据库系统是关系数据库系统与面向对象数据模型的结合。它保持了关系数据库系统的非过程化数据存取方式和数据独立性，继承了关系数据库系统已有的技术，支持

原有的数据管理，又能支持 OO 模型和对象管理。

有关面向对象数据模型的核心概念以及对象关系数据系统，我们将在第 15 章讨论。

4. XML 数据模型

随着互联网的迅速发展，Web 上各种半结构化、非结构化数据源已经成为重要的信息来源，XML 已成为网上数据交换的标准和数据界的研究热点。人们研究和提出了多种 XML 数据模型，还没有公认的统一的 XML 数据模型。W3C 已经提出的有：XML Information Set、Xpath1.0 Data Model、DOM model 和 XML Query Data Model。这四种模型都采用树结构。在这些模型中，XML Query Data Model 是较为完全的一种。当前，DBMS 产品都扩展了对 XML 的处理、存储 XML 数据、支持 XML 和关系数据之间的相互转换，我们将在第 16 章介绍 XML 数据库。

13.3.2 数据库技术与其他相关技术相结合

数据库技术与其他学科的内容相结合，是数据库技术的一个显著特征，随之也涌现出以下各种新型的数据库系统（如图 13.2 所示）。

- 数据库技术与分布处理技术相结合，出现了分布式数据库系统；
- 数据库技术与并行处理技术相结合，出现了并行数据库系统；
- 数据库技术与人工智能技术相结合，出现了知识库系统和主动数据库系统；
- 数据库技术与多媒体技术相结合，出现了多媒体数据库系统；
- 数据库技术与模糊技术相结合，出现了模糊数据库系统等；
- 数据库技术与移动通信技术相结合，出现了移动数据库系统等；
- 数据库技术与 Web 技术相结合，出现了 Web 数据库等。

这里仅仅举 2 个例子，描述数据库技术如何吸收、结合其他计算机技术，从而形成了数据库领域新的分支和研究课题，极大地丰富和发展了数据库技术。

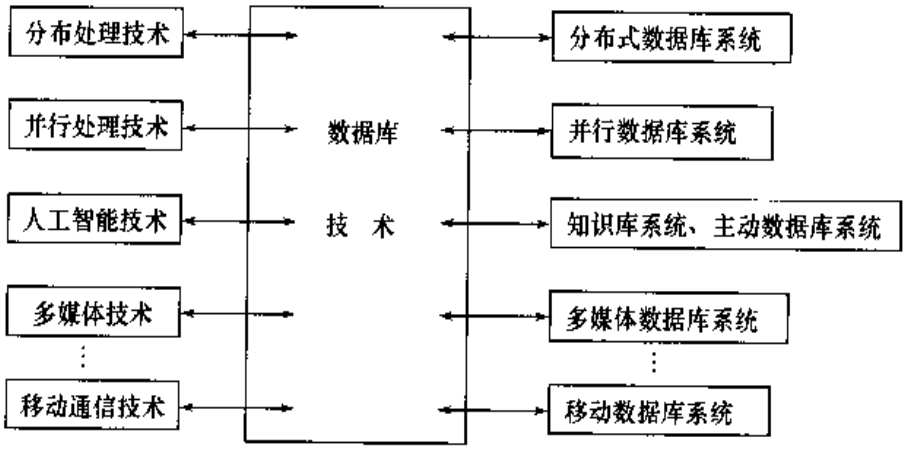


图 13.2 数据库技术与其他计算机技术的相互渗透

一、并行数据库系统

并行数据库系统是在并行机上运行的具有并行处理能力的数据库系统。并行数据库系统是数据库技术与并行计算技术相结合的产物。

近十几年来，数据库应用已经从商业事务处理迅速拓展到诸如超大型规模数据库检索、数据仓库、OLAP 联机数据分析、数据挖掘等应用领域。这些应用领域的特点是数据量巨大（TB 甚至 PB 级）、查询计算复杂，对数据库系统的处理能力提出了非常高的要求，这些应用需求直接驱动了新一代高性能数据库系统——并行数据库系统的研制。

随着微处理机技术和磁盘阵列技术的进步，并行计算机的发展十分迅速，出现了像 Sequent、Tandem、Teradata 和曙光机这样一些商品化的并行计算机系统。这些系统使用数个、数十个、甚至成百上千个廉价的微处理机协同工作，大大提高了性价比；而且，更为吸引人的是，这些系统由于广泛地采用了磁盘阵列技术，增加了 I/O 的带宽，能有效地解决应用中的 I/O 瓶颈问题。

并行处理技术与数据库技术的结合，具有潜在的可行性。这是因为关系数据库是元组的集合，数据库操作实际是集合操作，许多情况下可分解为一系列对子集的操作，具有潜在的并行性。并行计算技术利用多处理机并行处理产生的规模效益来提高系统的整体性能，为数据库系统提供了一个良好的硬件平台，形成了并行处理技术与数据库技术相结合的并行数据库新技术。

并行数据库系统研究一直以三种并行计算结构为基础，它们是：

- 共享内存（主存储器）结构（Shared_Memory）（以下简称 SM 结构）；
- 共享磁盘结构（Shared_Disk）（以下简称 SD 结构）；
- 无共享资源结构（Shared_Nothing）（以下简称 SN 结构）。

自并行数据库的概念提出以来，国外的许多研究机构相继研制出了各种结构平台上的并行数据库原型系统，例如加州 Berkeley 大学的 XPRS 系统、Colorado 大学的 Volcano 系统、Wisconsin 大学的 Gamma 系统等。这些系统探讨了并行数据库理论和实现中的若干重要问题，为并行数据库的进一步发展奠定了坚实的基础。与此同时，Teradata、Tandem 以及 Oracle、Sybase、Informix 等传统数据库厂商都正在或已经开发出自己的并行数据库系统。

国内中国人民大学、哈尔滨工业大学等单位开展了对并行数据库理论和系统实现技术的研究，开发了并行数据库管理系统。

二、主动数据库

主动数据库（Active Data Base）是在传统数据库基础上，结合人工智能技术和面向对象技术产生的数据库新技术。

相对于传统数据库的被动性而言，许多实际的应用领域，如计算机集成制造系统、管理信息系统、办公室自动化系统中常常希望数据库系统在紧急情况下能根据数据库的当前状态，主动适时地做出反应，执行某些操作，向用户提供有关信息。传统数据库系统是被动的

系统,它只能被动地按照用户给出的明确请求执行相应的数据库操作,很难充分适应这些应用的主动要求。主动数据库的主要目标是提供对紧急情况及时反应的能力。

主动数据库通常采用的方法是在传统数据库系统中嵌入 ECA,即事件 - 条件 - 动作规则,在某一事件发生时引发数据库管理系统去检测数据库当前状态,看是否满足设定的条件,若条件满足,便触发规定动作的执行。

为了有效地支持 ECA 规则,主动数据库的研究主要集中于解决以下问题。

1. 主动数据库的数据模型和知识模型。即如何扩充传统的数据库模型,使之能描述、存储、管理规则,适应主动数据库的要求。

2. 执行模型。即规则的处理和执行方式,是对传统数据库系统事务模型的发展和扩充。

3. 条件检测。是主动数据库系统实现的关键技术之一,由于条件复杂,如何高效地对条件求值对提高系统效率有很大的影响。

4. 事务调度。与传统数据库系统中的数据调度不同,它不仅要满足并发环境下的可串行化要求,而且要满足对事务时间方面的要求,研究对执行时间估计的代价模型。

5. 体系结构。目前,主动数据库的体系结构大多是在传统 DBMS 基础上,扩充事务管理部件和对象管理部件以支持执行模型和知识模型,并增加事件侦测部件、条件检测部件和规则管理部件。

6. 系统效率。系统效率是主动数据库研究中的一个重要问题,是设计各种算法和选择体系结构时应主要考虑的设计目标。

有关主动数据库的详细讨论,可参阅数据库丛书的《特种数据库》[①]。

13.3.3 面向应用领域的数据库新技术

数据库技术被应用到特定的领域中,出现了数据仓库、工程数据库、统计数据库、空间数据库、科学数据库等多种数据库(如图 13.3 所示),使数据库领域的应用范围不断扩大。

这些数据库系统都明显地带有某一领域应用需求的特征。由于传统数据库系统具有局限性,无法直接使用当前 DBMS 市场上销售的通用的 DBMS 来管理和处理这些领域内的数据对象,因而广大数据库工作者针对各个领域的数据库特征探索和研制了各种特定的数据库系统,取得了丰硕的成果,不仅为这些应用领域建立了可供使用的数据库系统,有的已实用化,而且为新一代数据库技术的发展作出了贡献。这里仅仅举 2 个例子加以说明。

一、工程数据库

工程数据库(Engineering Data Base)是一种能存储和管理各种工程设计图形和工程设计文档,并能为工程设计提供各种服务的数据库。传统的数据库能很好地存储规范数据和进

① 参见本章参考文献[11]。

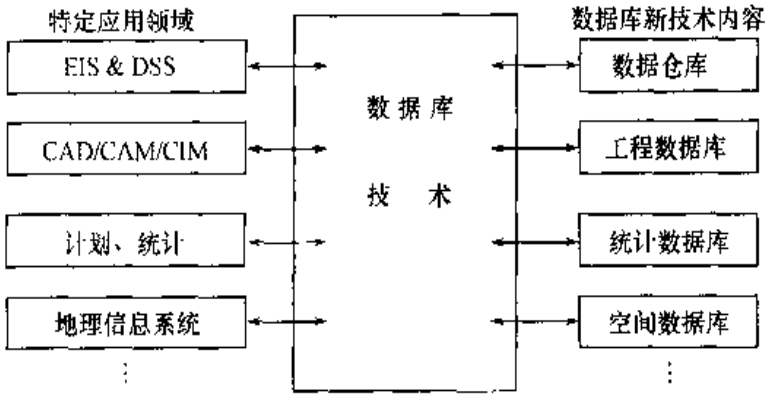


图 13.3 特定应用领域中的数据库技术

行事务处理，而在 CAD/CAM，CIM，CASE 等 CAx 的工程应用领域，对具有复杂结构和工程设计内涵的工程对象以及工程领域中的大量“非经典”应用传统的数据库则无能为力。工程数据库正是针对工程应用领域的需求而提出来的，目的是利用数据库技术对工程对象有效地加以管理，并提供相应的处理功能及良好的设计环境。

由于工程数据的数据结构复杂、相互联系紧密、数据存储量大，因此工程数据库管理系统的功能与传统 DBMS 有很大不同，主要应具有以下功能。

1. 支持复杂对象（如图形数据、工程设计文档）的表示和处理；
2. 可扩展的数据类型；
3. 支持复杂多样的工程数据的存储和集成管理；
4. 支持变长结构数据实体的处理；
5. 支持工程长事务和嵌套事务的并发控制和恢复；
6. 支持设计过程中多个不同数据版本的存储和管理；
7. 支持模式的动态修改和扩展；
8. 支持多种工程应用程序等。

在工程数据库的设计过程中，由于传统的数据模型难以满足工程应用的要求，需要运用当前数据库研究中一些新的模型技术，如扩展的关系模型、语义模型、面向对象的数据模型等。

目前的工程数据库的研究和开发虽然已取得了很大的成绩，但要全面达到应用所要求的目标仍有待进一步努力。

二、空间数据库

空间数据是用于表示空间物体的位置、形状、大小和分布特征等诸方面信息的数据，适用于描述所有二维、三维和多维分布的关于区域的现象。

空间数据的特点是不仅包括物体本身的空间位置及状态信息，还包括表示物体的空间关系（即拓扑关系）的信息。属性数据为非空间数据，用于描述空间物体的性质，对空间物体进行语义定义。

空间数据库系统(Spatial Data Base System)是描述、存储和处理空间数据及其属性数据的数据库系统。

空间数据库是随着地理信息系统(Geographical Information System)的开发和应用而发展起来的数据库新技术。目前,空间数据库系统不是独立存在的系统,它和应用紧密结合,大多数作为地理信息系统的基础和核心的形式出现。

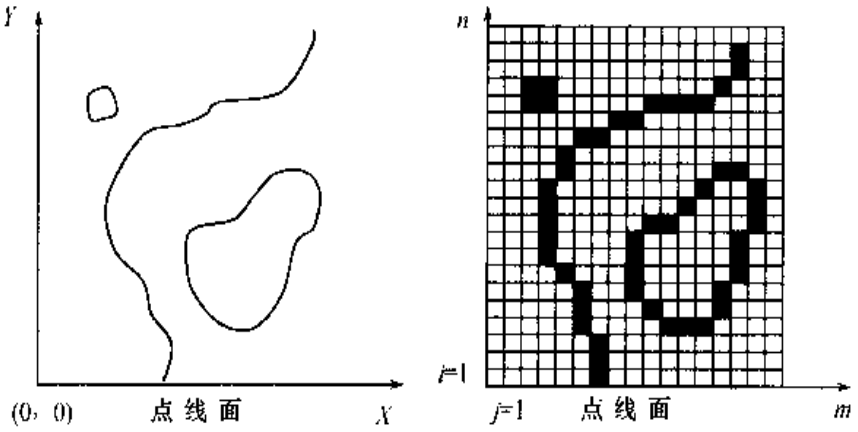
空间数据库的研究始于20世纪70年代的地图制图与遥感图像处理领域,其目的是为了有效地利用卫星遥感资源迅速绘制出各种经济专题地图。由于传统数据库在空间数据的表示、存储、管理和检索上存在许多缺陷,从而形成了空间数据库这一新的数据库研究领域,它涉及计算机科学、地理学、地图制图学、摄影测量与遥感、图像处理等多个学科。

空间数据库技术研究的主要内容如下。

1. 空间数据模型

空间数据模型是描述空间实体和空间实体关系的数据模型,一般来说可以用传统的数据模型加以扩充和修改来实现,也可以用面向对象的数据模型来实现。

空间数据库中常用的空间数据结构有矢量数据结构和栅格数据结构两种(如图13.4所示)。



(a) 矢量数据结构　　(b) 栅格数据结构

图13.4　空间数据结构

2. 空间数据查询语言

空间数据查询包括位置查询、空间关系查询和属性查询。前两种查询是空间数据库特有的,基本方式有以下几种。

- 面-面查询:例如查询与某一面状物A相邻的多边形;
- 线-线查询:例如查询某一河流A的所有支流;
- 点-点查询:例如查询两点之间的距离;
- 线-面查询:例如查询铁路A穿过的所有城镇;
- 点-线查询:例如查询某一河流A(线)上的所有桥梁(点);

• 点－面查询：例如查询某一城市 A 地图中的所有医院(点)。

空间数据查询语言是为了正确表达以上查询请求。

3. 空间数据库管理系统

空间数据库管理系统的主要功能是，提供对空间数据和空间关系的定义和描述，提供空间数据查询语言，实现对空间数据的高效查询和操作，提供对空间数据的存储和组织，提供对空间数据的直观显示等。

目前以空间数据库为核心的地理信息系统的应用已经从解决道路、输电线路等基础设施的规划和管理，扩展到更加复杂的领域。

面向特定领域的数据库系统，也称为特种数据库系统，还有很多，就不一一介绍了。

人们知道，推动数据库技术前进的原动力是应用需求和硬件平台的发展。正是这些应用需求的提出，特种数据库系统的研究，推动了新一代数据库技术的产生和发展。而新一代数据库技术也首先在这些特种数据库中发挥了作用，得到了应用。

从这些特种数据库系统的实现情况来分析，可以发现它们虽然采用不同的数据模型，但都带有 OO 模型的特征。具体实现时，有的是对关系数据库系统进行扩充，有的则是重新进行设计和开发。

13.4　数据库技术发展趋势

为了更好地把握数据库技术发展方向，国际上资深的数据库专家自 1988 年起每隔数年就聚在一起，对数据库研究现状加以分析和评价，指出存在的问题，提出未来值得关注的问题和研究领域。迄今为止，这样的会议已经举办了 6 次，最近的一次于 2003 年 5 月在美国麻省的 Lowell 举行。在每次会议上，专家们都做了充分的讨论甚至争论，提出了一些重要的观点，并写出详细的总结报告。本节结合最近的这次会议，简单介绍数据库研究的现状和发展趋势。有兴趣的读者可以参考本章参考文献[1]～[7]。

数据、应用需求和计算机相关技术是推动数据库发展的 3 个主要动力或 3 个重要因素，专家们分析了数据、应用需求和计算机相关技术的发展和变化。

一、信息特性和来源的变化

Internet、Web、科技、电子商务等已经成为信息和信息处理需求的重要来源。另一个来源是廉价的微型传感器，大量物体通过微型传感器实时报告它们的位置和状态。采集这些信息，可以用来监控这些物体。另一个越来越重要的来源是自然科学，特别是物理学、生物学、保健科学和工程领域，这些领域产生了大量复杂的数据集。

因此，数据库领域已经从获取、组织、存取、分析和恢复结构化数据扩展到文本、时间、空间、声音、图形图像、视频等多媒体数据，HTML、XML 等等非结构化和半结构化数据，还有程序数据、流数据和队列数据等等。

数据类型的多样化、处理这些数据的方法复杂化以及数据量越来越巨大是当前数据库面临的重要挑战。

是在原来的 DBMS 系统中增加对复杂数据类型的存储和处理功能，将新的结构移植到传统的架构上，还是应该重新思考 DBMS 基本架构，是当前学术界要研究的问题。

二、应用领域的变化

应用领域变化中，Internet 是当前最主要的驱动力。在 Internet 应用环境下所有应用已经从企业内部扩展为跨企业间的应用，需要 DBMS 对信息安全和信息集成提供更有力的保障和支持。这也就对 DBMS 研究团体提出了许多新的待解决的问题。

另一个重要的应用领域是科学研究领域，如物理、生物、生命科学和工程学。这些研究领域产生大量复杂的数据，需要比目前数据库产品所能提供的更高级的支持，同时也需要信息集成机制。此外，还需要对数据分析器产生的数据进行管理，需要存储有序数据和对它们查询（如时间序列、图像分析、网格计算和地理信息），需要世界范围内数据网格的集成。

三、相关技术的发展

另一个推动数据库研究发展的动力是相关技术的成熟。例如：在过去的十年中，数据挖掘技术已成为数据库系统的重要组成部分；Web 搜索引擎使得信息检索成了不可缺少的应用，这个检索技术也需要与经典的数据库搜索技术结合；许多人工智能的研究领域，也产生了能够与数据库技术结合的技术，这些技术使我们能够处理语音、自然语言、不确定性推理、机器学习等问题。

除了这些新的信息管理的挑战，在传统的 DBMS 技术方面也面临着巨大变化，数据模型、存取方法、查询算法，并发控制、恢复、查询语言以及用户操作界面等课题，虽然过去都有了较充分的研究，然而技术的发展一直在改变着规则。例如，磁盘和内存容量变得越来越大，成本却变得越来越便宜，访问时间和带宽也得到了改善，但是改善的速度却不及容量和成本的改善速度。这些变化要求我们对传统数据库存储管理和查询处理算法重新加以评估。另外，处理器高速缓存（cache）有了爆炸性增长，并且增加了层次，这就要求 DBMS 能够充分利用高速缓存来改进实现算法。

Lowell 报告[7]认为，许多影响信息管理的新应用、技术趋势、以及相关领域的协作等问题，需要一个全新的信息管理架构并需重新考虑信息存储、组织、管理和访问等方面的问题。

四、当前若干研究热点

Lowell 报告认为，信息管理仍然是大多数复杂软件系统的关键部分。同时给出了当前数据库研究领域的重要研究热点，包括文本、数据、代码、数据流的集成，异构数据源的信息融合，传感器数据和传感器网络，多媒体查询，不确定数据推理，个性化信息服务和信息处理，针对信息关联性的无人监督数据的挖掘技术，可信系统与信息隐私，DBMS 的自适应、自管理和自修复，新型用户界面，数据的永久保存，永恒的查询优化等等。限于篇幅，下面只列出几个方面，有兴趣的读者请参阅本章参考文献[1]、[9]以及最新的重要会议（如 SIGMOD、

VLDB、ICDE 等)论文。

1. 信息集成

随着 Internet 的飞速发展,Web 迅速成为全球性的分布式计算环境,Web 上有极其丰富的数据资源。如何获取 Web 上的有用数据并加以综合利用,成为一个广泛关注的研究领域。Web 数据源具有不同的数据类型(数据异构)、不同的模式结构(模式异构)、不同的语义内涵(语义异构);此外,Web 数据源还具有分布分散、动态变化、规模巨大等特点。这些都造成了 Web 信息集成与传统的异构数据库集成非常不同这个难题。

自 20 世纪 90 年代以来,数据库界在 Web 数据集成方面开展了大量研究。研究包括信息集成的数据仓库方法和 Wrapper/Mediator 方法,Web 信息检索技术,本体和 Semantic Web,Web 信息集成系统架构等,但是问题远没有解决。

2. 移动数据管理

研究移动计算环境中的数据管理技术,已成为目前一个新的方向,即嵌入式移动数据库技术。移动计算环境指的是具有无线通信能力的移动设备及其上运行的相关软件所共同构成的计算环境。移动计算环境使人们可以随时随地访问任意所需的信息。移动计算环境具有其鲜明的特点,主要包括:移动性和位置相关性、频繁的通信断接性、带宽多样性、网络通信的非对称性、移动设备的资源有限性等。

在嵌入式移动数据库系统中需要考虑与传统计算环境下不同的问题,如对断接操作的支持、对跨区长事务的支持、对位置相关查询的支持、对查询优化的特殊考虑以及对提高有限资源的利用率的考虑等。嵌入式移动数据库的关键技术主要包括:数据复制/缓存技术、移动事务处理技术、移动查询处理、服务器数据源的数据广播、移动用户管理、数据的安全性等。

3. 网格数据管理

简单地讲,网格是把整个网络整合成一个虚拟的、巨大的超级计算环境,实现计算资源、存储资源、数据资源、信息资源、知识资源、专家资源的全面共享。其目的是解决多机构虚拟组织中的资源共享和协同工作问题。

网格环境下的数据管理目标是,保证用户在存取数据时无需知道数据的存储类型(如数据库、文档、XML)和位置。涉及的问题包括:如何联合不同的物理数据源,抽取元数据构成逻辑数据源集合;如何制定统一的异构数据访问的接口标准;如何虚拟化分布的数据源等。

目前,数据网格研究的问题之一是,如何在网格环境下存取数据库,提供数据库层次的服务,因为数据库显然应该是网格中十分宝贵而巨大的数据资源。数据库网格服务不同于通常的数据库查询,也不同于传统的信息检索,需要将数据库提升为网格服务,把数据库查询技术和信息检索技术有机结合,提供统一的基于内容的数据库检索机制和软件。

此外,还有信息网格中个性化服务、信息安全性和语义 Web 的研究。

4. 传感器数据库技术

传感器网络由大量的低成本的设备组成,用来测量诸如目标位置、环境温度等数据。每

个设备都是一个数据源，将会提供重要的数据，这就产生了新的数据管理需求。

由于大量传感器设备具有移动性、分散性、动态性和传感器资源的有限性等特点，传感器数据库系统需要解决许多新的问题。例如，传感器数据的表示和传感器查询的表示和执行，在整个传感器网络中查找信息时，应该尽可能将计算分布到各个结点上以提高性能。传感器资源的有限性需要研究传感器节点上的查询处理新技术。在传感器网络上，查询执行要适应随时迅速变化的情况，例如，传感器从网络上分离或者消失，查询计划也需要随之变化，这在当前的数据库系统中是没有的。传感器也提出了处理更复杂的信息处理的需求，一个常见的情况是，传感器没有完全校准，某个传感器的准确值要依据其他传感器的信号来求证。更复杂的情况是，传感器数据处理的目标可能要从多个低级别的信号演绎出一个高级别的信息。例如，需要综合温度、声音、振动等多种传感器信号来定位附近的一个人。

5. DBMS 的自适应管理

随着 RDBMS 复杂性的增强以及新功能的增加，对于 DBA 的技术需求、熟练 DBA 的薪水支出都在大幅度增长，从而导致企业人力成本的支出迅速增加。随着关系数据库规模和复杂性的增加，系统调整和管理的复杂性相应增加。基于上述原因，数据库系统自调优和自管理工具的需求增加，对数据库自调优和自管理的研究也逐渐成为关注的热点。

数据库系统的最终目标是“没有可调部分”，即是说，所有的调整均由 DBMS 自动完成。它可以依据缺省的规则，如响应时间和吞吐率的相对重要性做出选择，也可以依据用户的需要制定规则。因此，建立能够清楚描述用户行为和工作负载的更完善的模型，是这一领域取得进展的先决条件。除了不需要手工调整，DBMS 还需要能够发现系统组件内部及组件之间的故障，辨别数据冲突，侦查应用失败，并且做出相应的处理。这就要求 DBMS 具有更强的适应性和故障处理能力。

学术界与工业界都在努力，有的数据库厂商已将部分研究成果转化到产品中去。

13.5 小 结

数据库系统已经发展成为一个大家族，本章以数据模型、新技术内容、应用领域为三条主线，概要地回顾了数据库技术发展的三个阶段，阐述了数据库技术的发展及其相互关系，达到纲举目张。

数据库技术的核心是数据管理。随着新应用领域不断涌现，数据对象趋于多样化，数据库工作者也应该不断拓宽数据库的研究领域，在众多新领域中勇敢地承担起其中的数据管理研究开发任务。数据库工作者应该具有这样的意识，哪里有数据，哪里就有数据管理的任务，哪里就有数据库研究和开发的课题。面对新的挑战，我们要继承数据库技术和其他技术相结合的优良传统，努力探索新的途径、新的方法、新的技术，来提高和改善对数据和信息的使用。

最后，对我国数据库工作者来说可能面临更大的压力和挑战，这就是：我们需要在研究与成果转化两条战线工作，需要在继承和创新两个方面提高。

习　题

1. 请阅读本章参考文献[1]。
2. 试述数据库技术的发展过程。数据库技术发展的特点是什么？
3. 试述第一、二代数据库系统的主要成就。
4. 第三代数据库系统的主要特征是什么？
5. 试述数据模型在数据库系统发展中的作用和地位。
6. 请用实例阐述数据库技术与其他学科的技术相结合的成果。

本章参考文献

1　王珊主编. 数据库与信息系统：研究与挑战(1988－2003研究报告). 北京：高等教育出版社，2005

（自1988年起，每隔几年，国际上一些资深的数据库专家就会聚集一堂，探讨数据库的研究现状、存在的问题和未来需要关注的新的技术焦点。迄今为止，这样的会议已经举办了6次，分析这6次会议的总结报告，能够帮助我们清晰地把握数据库技术进展的脉络，从中也体会到数据库专家们当年的远见卓识。

为了让更多的人分享国际资深专家的工作成果，我们将6次会议的总结报告做了编译和整理。本书分为上下两篇，上篇是对会议报告的编译整理，下篇是会议报告的英语原文，供读者选用。）

2　Bernstein P, Dayal U, DeWitt DJ, Gawlick D, Gray J, Jarke M, Lindsay BG, Lockemann PC, Maier D, Neuhold EJ, Reuter A, Rowe LA, Schek H, Schmidt JW, Schrefl M, Stonebraker ML. Future directions in DBMS research——The laguna beachparticipants. SIGMOD Record, 1989, 18(1): 17～26.

3　Silberschatz A, Stonebraker M, Ullman JD. Database systems: Achievements and opportunities. CACM, 1991, 34(10): 110～120.

4　Silberschatz A, Stonebraker M, Ullman JD. Database research, achievements and opportunities into the 21st century. SIGMOD Record, 1996, 25(1): 52～63.

5　Silberschatz A, Zdonik SB. Strategic directions in database systems——Breaking out of the box. ACM Computing Surveys, 1996, 28(4): 764～778.

6　Bernstein P, Brodie ML, Ceri S, DeWitt DJ, Franklin MJ, Garcia-Molina H, Gray J, Held G, Hellerstein JM, Jagadish HV, Lesk M, Maier D, Naughton JF, Pirahesh H, Stonebraker M, Ullman JD. The asilomar report on database research. SIGMOD Record, 1998, 27(4): 74～80.

7　Abiteboul S, Agrawal R, Bernstein P, Carey M, Ceri S, Croft B, DeWitt D, Franklin M, Molina H, Gawlick D, Gray J, Haas L, Halevy A, Hellerstein J, Ioannidis Y, Kersten M, Pazzani M, Lesk M, Maier D, Naughton J, Schek H, Sellis T, Silberschatz A, Stonebraker M, Snodgrass R, Ullman J, Weikum G, Widom J, Zdonik S. The Lowell Database Research Self-Assessment Meeting, Lowell Massachusetts, 2003. http://research. microsoft.

com/ ~ gray/lowell

8 CRA Conference on "Grand Research Challenges" in Computer Science and Engineering, http://www.cra.org/Activities/grand.challenges/.

9 孟小峰,周龙骧,王珊.数据库技术发展趋势.软件学报,卷 15(12):1822 - 1836,2004

10 李昭原主编.数据库技术新进展.北京:清华大学出版社,1997

11 何新贵,唐常杰,李霖.特种数据库技术.数据库技术丛书之一,北京:科学出版社,1999

(本书主要包括时态数据库技术、移动数据库技术、主动数据库技术和模糊数据库技术等,每部分都讲解了各领域的基本理论和处理技术,以及国内外有关学者们在相应领域中的最新研究成果等。)

* 第十四章 分布式数据库系统

由于计算机网络通信的迅速发展以及地理上分散的公司、团体和组织对数据库更为广泛的应用的需求，20 世纪 80 年代，在集中式数据库系统成熟技术的基础上产生和发展了分布式数据库系统(Distributed Database System)。分布式数据库系统是数据库技术和网络技术两者相互渗透和有机结合的结果。

这一章介绍分布式数据库系统的基本概念，由于分布式数据库系统比集中式数据库系统更复杂，在有限的篇幅中只能论述最基本的概念和技术，作为今后进一步学习分布式数据库系统的基础。

14.1 概 述

14.1.1 分布式数据库系统

什么样的一个数据库系统才算是分布式数据库系统呢？一个粗略的定义是：

“分布式数据库由一组数据组成，这些数据物理上分布在计算机网络的不同结点(亦称场地)上，逻辑上是属于同一个系统”。

这个定义强调了下面两点：

(1) 分布性。数据库中的数据不是存储在同一场地上，更确切地讲，不存储在同一计算机的存储设备上，这就可以和集中式数据库相区别。

(2) 逻辑整体性。这些数据逻辑上是互相联系的，是一个整体(逻辑上如同集中数据库)。这就可以和分散在计算机网络不同结点上的那些集中式数据库或文件的集合相区别。后者各结点的数据之间没有内在的逻辑联系。所以在讨论分布式数据库时就有了全局数据库(逻辑上)和局部数据库(物理上)的概念。

但是，这个定义仍然是不精确的，下面来分析两个例子。

[例 1] 如图 14.1 所示，这个系统中有 3 台服务器，每台服务器有自己的数据库系统，3 台服务器之间通过网络相连，每台服务器有自己的若干客户机。用户可以通过客户机对本

地服务器中的数据库执行某些应用(称之为局部应用),也可以通过客户机对两个或两个以上结点中的数据库执行某些应用(称之为全局应用或分布应用)。这样的系统是分布式数据库系统,而不支持全局应用的系统不能称为分布式数据库系统。一个典型的全局应用的例子是银行转账。这个应用要求从一个分行的账户(设在 DB1 数据库)中转移若干金额到另一个分行的账户(设在 DB3 数据库)中去,因此要同时更新两个结点上的数据库。

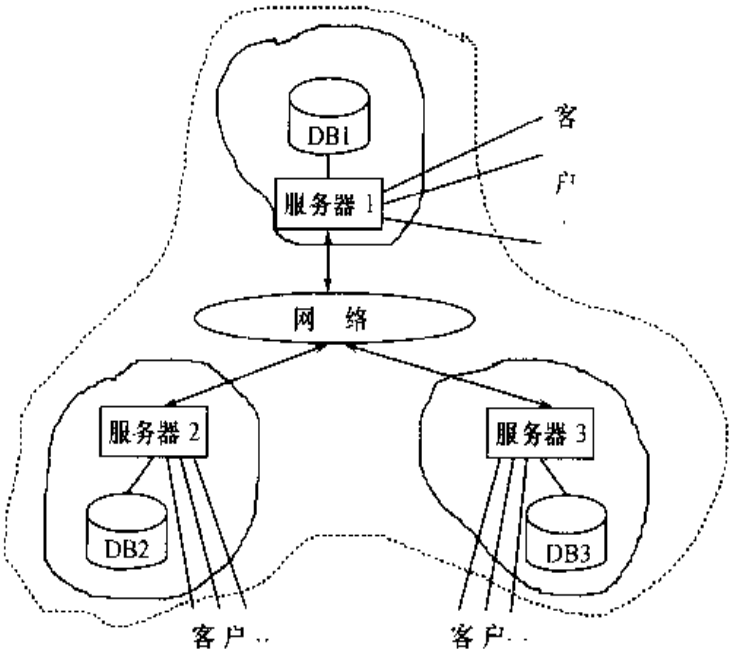


图 14.1 一个分布式数据库系统

这个例子说明,区分一个系统是若干集中式数据库的简单连网还是分布式数据库系统的技术要点在于,系统是否支持全局应用。所谓全局应用是指应用中涉及两个或两个以上结点的数据库。它和只存取本结点数据库的应用不同,和只存取另一个结点(或远程结点)数据库的应用也是不同的。支持全局应用的技术要复杂困难得多。在图 14.1 中用实线示意局部数据库及局部应用,用虚线示意全局数据库及全局应用。

[例 2] 如图 14.2 所示,这个系统的配置和图 14.1 的不同点是:3 个数据库通过后台服务器和网络相连。服务器执行数据库管理功能。所有的应用都由客户机处理。尽管发出这些应用的客户可以分布在不同的场地,数据库也物理地分布在各个不同的服务器上,但这样的系统不是分布式数据库系统而是一个多处理机系统(SN 并行结构),其原因是没有局部应用,即每个后台服务器不能由它本身来执行自己的局部应用,数据的物理分布不是对应用观点而言的分布,而分布式数据库不仅要求数据的物理分布,而且要求这种分布是面向处理、面向应用的。

总结上面两个例子,可以得出分布式数据库的一个更确切的定义:

分布式数据库是由一组数据组成的,这组数据分布在计算机网络的不同计算机上,网络

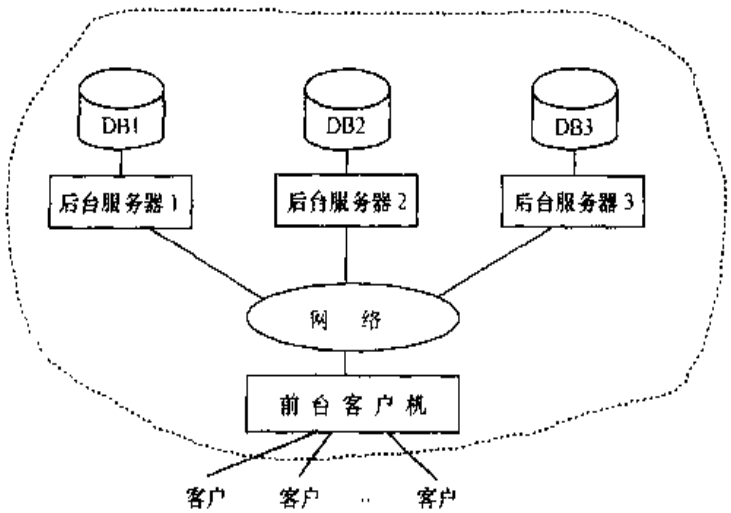


图 14.2 一个多处理机系统(SN 并行结构)

中的每个结点具有独立处理的能力(称为场地自治),可以执行局部应用。同时,每个结点也能通过网络通信子系统执行全局应用。

这个定义比前一个定义更强调了场地**自治性**以及自治场地之间的**协作性**。这就是说,每个场地是独立的数据库系统,它有自己的数据库、自己的用户、自己的 CPU,运行自己的 DBMS,执行局部应用,具有高度的自治性。同时各个场地的数据库系统又相互协作组成一个整体。这种整体性的含义是,对于用户来说,一个分布式数据库系统逻辑上看如同一个集中式数据库系统一样 ,用户可以在任何一个场地执行全局应用。

14.1.2 分布式数据库系统的特点

分布式数据库系统是在集中式数据库系统技术的基础上发展起来的。它具有自己的特性和特征。集中式数据库的许多概念和技术,如数据独立性、数据共享和减少冗余度、并发控制、完整性、安全性和恢复等,在分布式数据库系统中都有了更加丰富的内容。

一、数据独立性

数据独立性是数据库技术追求的主要目标之一。在集中式数据库系统中,数据独立性包括两个方面:数据的逻辑独立性与数据的物理独立性。其含义是用户程序与数据的全局逻辑结构及数据的存储结构无关。

在分布式数据库系统中,数据独立性这一特性更加重要,并具有更多的内容。除了数据的逻辑独立性与物理独立性外,还有数据分布独立性亦称分布透明性(Distribution Transparency)。分布透明性指用户不必关心数据的逻辑分片,不必关心数据物理位置分布的细节,也不必关心重复副本(冗余数据)一致性问题,同时也不必关心局部场地上数据库支持哪种数据模型。分布透明性也可以归入物理独立性的范围。

分布透明性的优点是很明显的。有了分布透明性,用户的应用程序书写起来就如同数

据没有分布一样。当数据从一个场地移到另一个场地时不必改写应用程序。当增加某些数据的重复副本时也不必改写应用程序。数据分布的信息由系统存储在数据字典中。用户对非本地数据的访问请求由系统根据数据字典予以解释、转换和传送。

在集中式数据库系统中,数据独立性是通过系统的三级模式(外模式、模式、内模式)和它们之间的二级映像得到的。在分布式数据库系统中,分布透明性则是由于引入了新的模式和模式间的映像得到的。这将在 14.2.1 节中详细讨论。

二、集中与自治相结合的控制结构

数据库是多个用户共享的资源。在集中式数据库系统中,为了保证数据库的安全性和完整性,对共享数据库的控制是集中的,并由 DBA 负责监督和维护系统的正常运行。

在分布式数据库系统中,数据的共享有两个层次:

(1) 局部共享。即在局部数据库中存储局部场地上各用户的共享数据,这些数据是本场地用户常用的。

(2) 全局共享。即在分布式数据库系统的各个场地也存储供其他场地的用户共享的数据,支持系统的全局应用。

因此,相应的控制机构也具有两个层次:集中和自治。分布式数据库系统常常采用集中和自治相结合的控制机构。各局部的 DBMS 可以独立地管理局部数据库,具有自治的功能。同时,系统又设有集中控制机制,协调各局部 DBMS 的工作,执行全局应用。当然,不同的系统,集中和自治的程度不尽相同。有些系统高度自治,连全局应用事务的协调也由局部 DBMS、局部 DBA 共同承担,而不要集中控制,不设全局 DBA。有些系统则集中控制程度较高,场地自治功能较弱。

三、适当增加数据冗余度

在集中式数据库系统中,尽量减少冗余度是系统目标之一。其原因是,冗余数据不仅浪费存储空间,而且容易造成各数据副本之间的不一致性,为了保证数据的一致性,系统要付出一定的维护代价。减少冗余度的目标是用数据共享来达到的。

而在分布式数据库系统中却希望存储必要的冗余数据,在不同的场地存储同一数据的多个副本,其原因是:

1. 提高系统的可靠性、可用性

当某一场地出现故障时,系统可以对另一场地上的相同副本进行操作,不会因一处故障而造成整个系统的瘫痪。

2. 提高系统性能

系统可以选择用户最近的数据副本进行操作,减少通信代价,改善整个系统的性能。

但是,数据冗余会带来和集中式数据库系统中一样的问题。尤其是冗余副本之间数据不一致的问题是分布式数据库系统必须着力解决的问题。

一般地讲,增加数据冗余度方便了检索,提高了系统的查询速度、可用性和可靠性,但不

利于更新，增加了系统维护的代价。因此应在这些方面作出权衡，进行优化。

四、全局的一致性、可串行性和可恢复性

分布式数据库系统中各局部数据库应满足集中式数据库的一致性、并发事务的可串行性和可恢复性。除此以外还应保证数据库的全局一致性、全局并发事务的可串行性和系统的全局可恢复性。这是因为在分布式数据库系统中全局应用要涉及两个以上结点的数据，全局事务可能由不同场地上的多个操作组成。例如 14.1.1 节的[例 1]中银行转账事务包括两个结点上的更新操作。这样，当其中某一个结点出现故障，操作失败后如何使全局事务回滚呢？如何使另一个结点撤销(UNDO)已执行的操作(若操作已完成或完成一部分)或者不必再执行事务的其他操作(若操作尚未执行)？这些技术要比集中式数据库系统复杂和困难得多，分布式数据库系统必须解决这些问题。

14.2　分布式数据库系统的体系结构

这一节讨论满足 14.1 节中给出的分布式数据库系统特点的系统结构(包括模式结构和分布式数据库管理系统的结构)。同时进一步阐述分布式数据库系统的基本概念。

14.2.1　分布式数据库系统的模式结构

图 14.3 是分布式数据库系统一种模式结构的示意图，实际的系统并非都具有这种结构。在这个结构中各级模式的层次清晰，可以概括和说明分布式数据库系统的概念和结构。

图 14.3 的模式结构从整体上可以分为两大部分：下部是集中式数据库系统的模式结构，代表了各局部场地上局部数据库系统的基本结构；上部是分布式数据库系统增加的模式级别，其中包括：

(1) 全局外模式(Global External Schema)，它们是全局应用的用户视图，是全局概念模式的子集。

(2) 全局概念模式(Global Conceptual Schema)，它定义分布式数据库中数据的整体逻辑结构，使得数据如同没有分布一样。全局概念模式中所用的数据模型应该易于向其他模式映像，通常采用关系模型。这样，全局概念模式包括一组全局关系的定义。

(3) 分片模式(Fragmentation Schema)，每一个全局关系可以分为若干不相交的部分，每一部分称为一个片段(Fragment)。分片模式定义片段以及全局关系到片段的映像。这种映像是一对多的，一个全局关系可对应多个片段，而一个片段只来自一个全局关系。

(4) 分布模式(Allocation Schema)，片段是全局关系的逻辑部分，一个片段在物理上可以分配到网络的不同结点上。分布模式定义片段的存放结点。分布模式的映像类型确定了分布式数据库是冗余的还是非冗余的。若映像是一对多的，即一个片段可分配到多个结点上存放，则是冗余的分布式数据库，若映像是一对一的，则是非冗余的分布式数据库。

根据分布模式提供的信息，一个全局查询可分解为若干子查询，每一子查询要访问的数据属于同一场地的局部数据库。

由分布模式到各局部数据库的映像（映像4）把存储在局部场地的全局关系或全局关系的片段映像为各局部概念模式（Local Conceptual Schema），局部概念模式采用局部场地的DBMS所支持的数据模型。

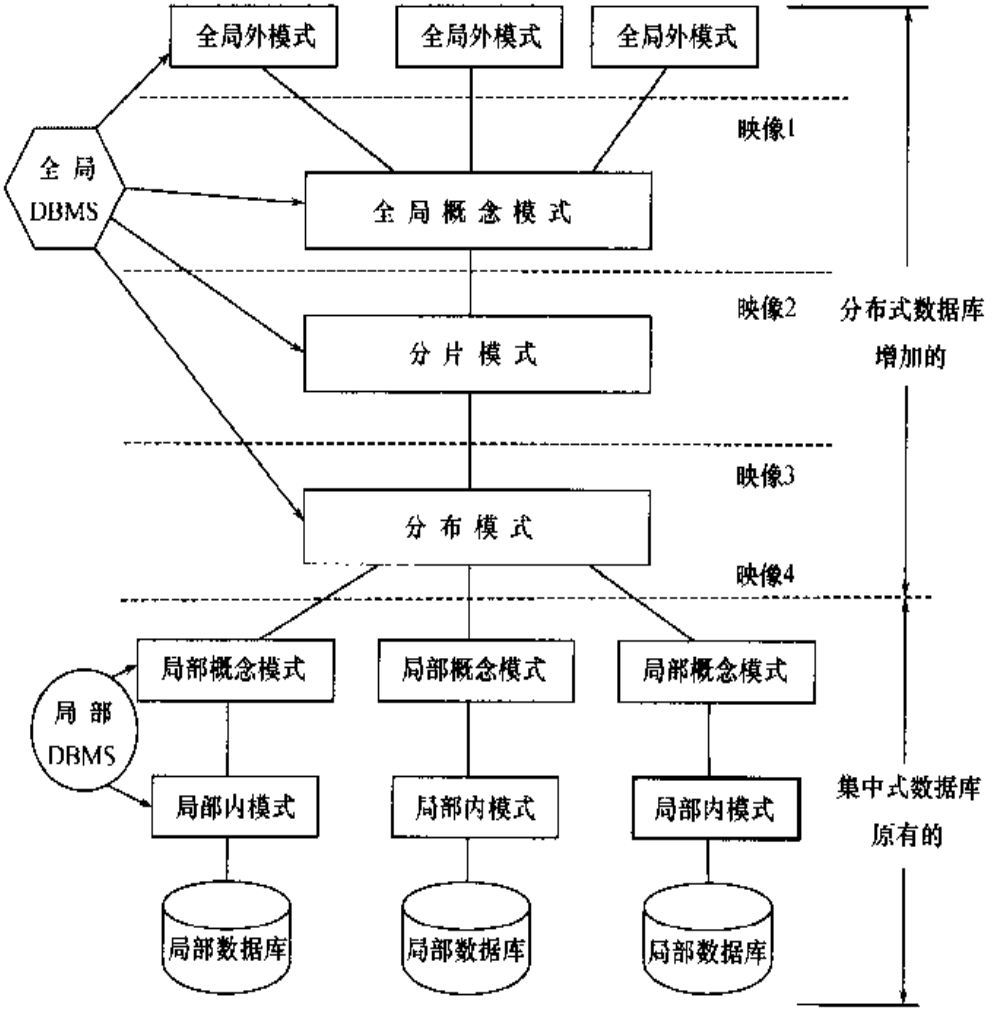


图14.3 分布式数据库系统的模式结构

分片模式和分布模式均是全局的，分布式数据库系统中增加的这些模式和相应的映像使分布式数据库系统具有了分布透明性。

14.2.2 数据分片

将数据分片，使数据存放的单位不是关系而是片段，这既有利于按照用户的需求较好地组织数据的分布，也有利于控制数据的冗余度。

分片的方式有多种，水平分片和垂直分片是两种基本的分片方式，混合分片和导出分片是较复杂的分片方式。

水平分片是指按一定的条件将关系按行(水平方向)分为若干不相交的子集,每个子集为关系的一个片段。

垂直分片是指将关系按列(垂直方向)分为若干子集。垂直分片的诸片段必须能够重构原来的全局关系,即可以用连接的方法恢复原关系,因此垂直分片的诸片段通常都包含关系的码。

导出分片是指导出水平分片,即水平分片的条件不是本身属性的条件而是其他关系的属性的条件。例如学生选课关系 SC(Sno,Cno,Grade),若不是按照学号或课程号或成绩的某类条件分片,而是按照学生年龄 >18 岁和≤18 岁来分片(学生年龄是学生关系 Student 的属性),则这类分片称导出分片,可以用 SQL 语句来表达这两个片段。

年龄 >18 岁的学生选课片段 SC_A(Sno,Cno,Grade)由下面的查询结果组成:

```
SELECT Sno,Cno,Grade
FROM S,SC
WHERE S.Sno = SC.Sno AND S.Sage > 18;
```

年龄≤18 岁的片段 SC_B 由下面的查询结果组成:

```
SELECT Sno,Cno,Grade
FROM S,SC
WHERE S.Sno = SC.Sno AND S.Sage≤18;
```

混合分片是指按上述三种分片方式得到的片段继续按另一种方式分片。例如,先按水平分片得到的某一片段再进行垂直分片(如图 14.4(a)所示),或者按垂直分片得到的某一片段又按水平分片(如图 14.4(b)所示)方式继续分为若干片段。

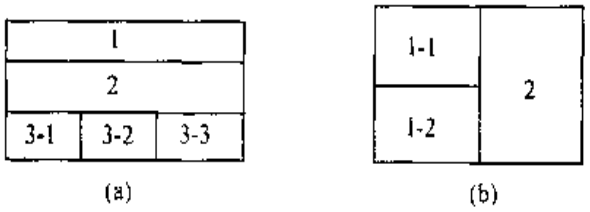


图 14.4　混合分片

无论哪种分片方式都应满足以下条件:

(1) 完全性。一个全局关系中的数据必须完全地划分为若干片段,不允许某些数据属于全局关系但不属于任何一个片段。

(2) 不相交性。不允许一个全局关系的某些数据既属于该全局关系的某一个片段又属于该全局关系的另一个片段(垂直分片中的码属性除外)。

(3) 可重构性。可以由片段重构全局关系,对于垂直分片可以用连接操作重构全局关系。例如,R_1,R_2,R_3 为全局关系 R 的垂直分片,R 的码属性 A,则

$$R = R_1 \underset{R_1.A=R_2.A}{\bowtie} R_2 \underset{R_2.A=R_3.A}{\bowtie} R_3$$

对于水平分片，可以用并操作重构全局关系，例如，SC_A，SC_B 为 SC 的两个水平分片，则

$$SC = SC_A \cup SC_B$$

14.2.3 分布透明性

从图 14.3 的模式结构可以看到分布透明性包括：分片透明性、位置透明性和局部数据模型透明性。

分片透明性是分布透明性的最高层次。所谓分片透明性是指用户或应用程序只对全局关系进行操作而不必考虑关系的分片。当分片模式改变了，由于全局模式到分片模式的映像（映像 2），全局模式不变，应用程序不必改写，这就是分片透明性。

位置透明是分布透明的下一层次。所谓位置透明性是指，用户或应用程序不必了解片段的存储场地，当存储场地改变了，由于分片模式到分布模式的映像（映像 3），应用程序不必改变。同时，若片段的重复副本数目改变了，数据的冗余度改变了，用户也不必关心如何保持各副本的一致性，这就是重复副本的透明性。

局部数据模型透明性是指用户或用户程序不必了解局部场地上使用的是哪种数据模型，模型的转换以及数据库语言的转换均由映像 4 完成。

下面讨论一个简单的应用，用它来说明对于不同层次上的透明性，应如何编写应用程序。可以发现，透明性层次越高，应用程序的编写越简单。

[例 1] 设在分布式数据库系统中有全局关系 Student(Sno,Sname,Sdept,Sage)，Student 关系被划分为两个片段 S_A 和 S_B。S_A 代表理学院的学生，S_B 代表文学院的学生。S_A 存储在场地 1(Site1)，S_B 冗余地存储在场地 2 和场地 3 上（如图 14.5 所示）。

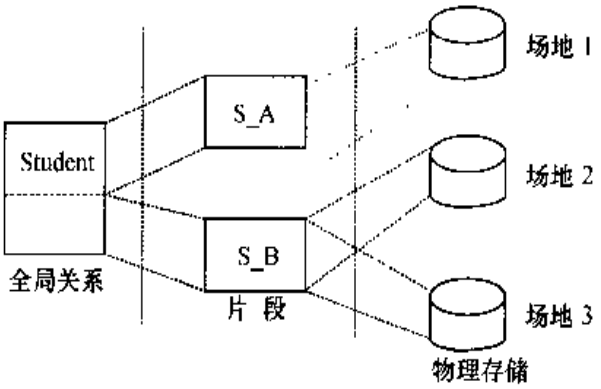


图 14.5 全局关系、片段及物理存储的关系

这里有一个查询，从终端读入一个学号，查找该学号的学生姓名、年龄，并把它们显示在屏幕上。

设应用程序是用嵌入 SQL 语句的 C 语言写的。这里略去了程序细节，仅给出查询部分的算法思想。

情况 1 若系统具有分片透明性，则

```
Scanf("% s",Snumber);                    /* 从终端读入学号到变量 Snumber 中 */
EXEC SQL SELECT Sname,Sage INTO :NAME,:AGE
```

```
                                        /* NAME,AGE 为程序变量 */
FROM Student                            /* 在全局关系 Student 中查找 */
WHERE Sno = :Snumber;
Printf("% s,% d",NAME,AGE);             /* 把 NAME,AGE 打印在屏幕上 */
```

情况 2 若系统具有位置透明性,但不具有分片透明性,则

```
Scanf("% s",Snumber);
EXEC SQL SELECT Sname,Sage INTO :NAME,:AGE
FROM S_A                                /* 先在片段 S_A 中查找 */
WHERE Sno = :Snumber;
If(! FOUND){
EXEC SQL SELECT Sname,Sage INTO :NAME,:AGE
FROM S_B                                /* 再在片段 S_B 中查找 */
WHERE Sno = :Snumber;
}
Printf("% s,% d",NAME,AGE);
```

情况 3 若系统只具有局部数据模型透明性,不具有位置透明性(当然也就不具有分片透明性),则

```
Scanf("% s",Snumber);
EXEC SQL SELECT Sname,Sage INTO :NAME,:AGE
        FROM S_A AT Site1               /* 先在场地 1 的片段 S_A 中查找 */
        WHERE Sno = :Snumber;
If(! FOUND){
EXEC SQL SELECT Sname,Sage INTO :NAME,:AGE
        FROM S_B AT Site2               /* 再在场地 2 的片段 S_B 中查找 */
        WHERE Sno = :Snumber ;          /* 也可以在场地 3 的片段 S_B 中查找 */
}
Printf("% s,% d",NAME,AGE);
```

因为 S_B 重复存放在场地 2 和场地 3,所以用户程序可以对场地 2,或场地 3 访问,用户可选择离他较近的某个场地进行访问,这样可以提高查询效率。

从这里看到,若没有分片透明性,用户必须了解分片的情况。若没有位置透明性,场地的选择由用户程序负责,用户不仅要了解分片的情况,还必须了解片段存放的位置,场地分布的远近,以及通信线路的质量。若系统具有位置透明性,则这些任务均由系统承担,并且当访问场地 2 时,若该线路繁忙或有了通信故障,则系统可以自动地改为访问场地 3,而这种优化工作用户程序是做不到的。

14.2.4 分布式数据库管理系统

分布式数据库管理系统(Distributed Data Management System,D-DBMS)是建立、管理和

维护分布式数据库的一组软件。这里给出一种 D-DBMS 的结构，分析它的主要成分和功能（如图 14.6 所示）。

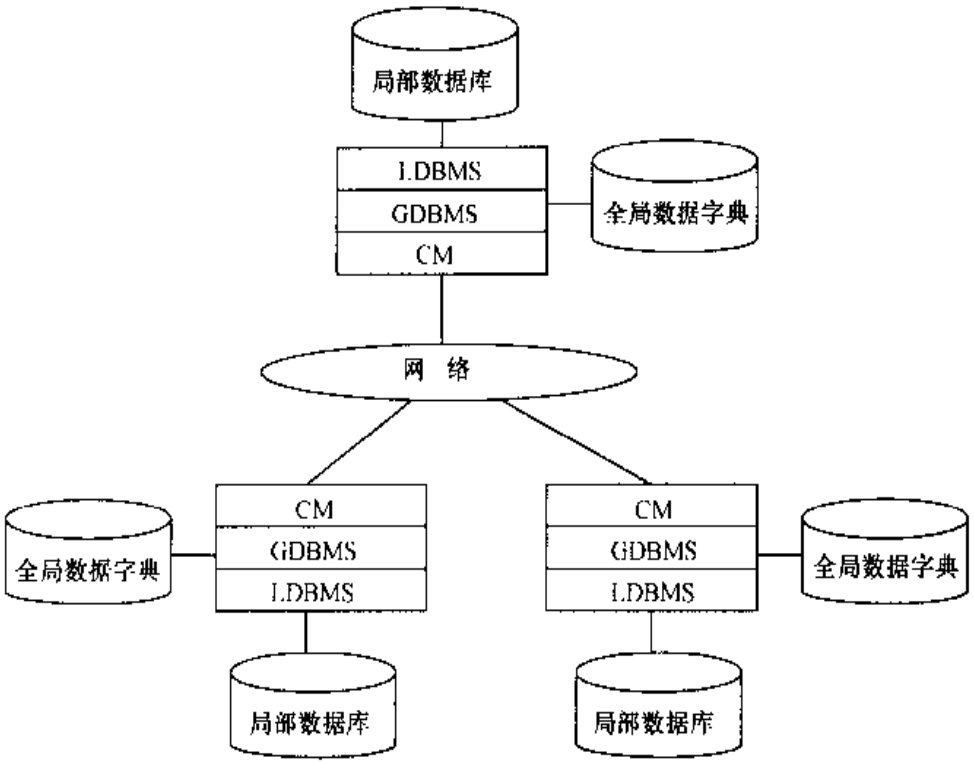


图 14.6 分布式数据库管理系统的结构

D-DBMS 由四部分组成：

（1）LDBMS（Local DBMS），局部场地上的数据库管理系统，其功能是建立和管理局部数据库，提供场地自治能力，执行局部应用及全局查询的子查询。

（2）GDBMS（Global DBMS），全局数据库管理系统，主要功能是提供分布透明性（如图 14.3 所示），协调全局事务的执行，协调各局部 DBMS 以完成全局应用，保证数据库的全局一致性，执行并发控制，实现更新同步，提供全局恢复功能等。

（3）全局数据字典（Global Data Directory，GDD），存放全局概念模式、分片模式、分布模式的定义以及各模式之间映像的定义，存放有关用户存取权限的定义，以保证用户的合法权限和数据库的安全性，存放数据完整性约束条件的定义，其功能与集中式数据库的数据字典类似。

（4）通信管理（Communication Management，CM），通信管理系统在分布式数据库各场地之间传送消息和数据，完成通信功能。

D-DBMS 功能的分割和重复以及不同的配置策略就导致了各种不同的体系结构。

一、按全局控制方式分类

1. 全局控制集中的 D-DBMS

这种结构的特点是全局控制成分 CDBMS 集中在某一结点上，由该结点完成全局事务的

协调和局部数据库转换等一切控制功能。全局数据字典只有一个,也存放在该结点上,它是GDBMS执行控制的主要依据。

这种结构的优点是控制简单,容易实现更新一致性,但由于控制集中在某一特定的结点上,不仅容易形成瓶颈而且系统较脆弱,一旦该结点出故障,整个系统就将瘫痪。

2. 全局控制分散的D-DBMS

这种结构的特点是全局控制成分GDBMS分散在网络的每一个结点上,全局数据字典也在每个结点上存放一份。每个结点都能完成全局事务的协调和局部数据库转换的控制功能,每个结点既是全局事务的参与者又是全局事务的协调者。图14.6就是这种体系结构。一般称这类结构为完全分布的D-DBMS。

它的优点是结点独立,自治性强,单个结点退出或进入系统均不会影响整个系统的运行,但是全局控制的协调机制和一致性的维护都比较复杂。

3. 全局控制部分分散的D-DBMS

这种结构是根据应用的需要将GDBMS和全局数据字典分散在某些结点上,是介于前两种情况的体系结构。

二、按局部DBMS的类型分类

区分不同D-DBMS的一个重要性质是:局部DBMS是同构的还是异构的。

同构和异构的级别可以有三级:硬件、操作系统和局部DBMS。其中最主要的是局部DBMS这一级,因为硬件和操作系统的不同将由通信软件处理和管理。所以,定义同构型D-DBMS为:每个结点的局部数据库具有相同的DBMS,如都是Oracle关系数据库管理系统,即使操作系统和计算机硬件并不相同;异构型D-DBMS为:各结点的局部数据库具有不同的DBMS,如有的是Oracle,有的是DB2,有的是IMS层次数据库管理系统。

异构型D-DBMS的设计和实现比同构型D-DBMS更加复杂。

14.3 查询处理和优化

分布式数据库系统中的查询处理较集中式数据库系统复杂,查询优化较集中数据库系统更重要,效果更显著。下面首先讨论一个实例,说明在分布环境下选择一个好的处理查询策略是多么重要,并了解查询处理和优化涉及的问题,然后讨论查询优化的一般目标和策略。

14.3.1 一个实例

下面是一个很有说服力的例子,说明存取策略优化的重要性。

- 数据库:简化了的供应商和零件数据库

 S(Sno,City)　　10^4 个元组,存放在场地A;

P(Pno,Color)　　10^5 个元组,存放在场地 B;

SP(Sno,Pno)　　16^6 个元组,存放在场地 A;

设每个关系的元组均为 100 字节长。

- 查询:求供应红色零件的、北京的供应商号

```
SELECT S. Sno
FROM S,P,SP
WHERE S. City ='北京' AND
        SP. Pno = P. Pno AND
        P. Color ='红色'
```

- 估算值(某些中间结果的元组数)

 红色零件数 = 10

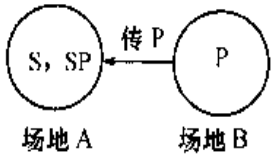
 北京供应商的装运单数 = 10^5

- 对通信系统的假定

 数据传输速度 = 10^4 字节/秒

 传输延迟 = 1 秒

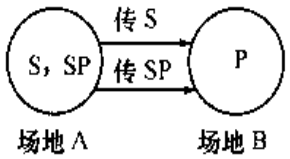
现在,考虑下面 6 种可能的查询存取策略,对各种策略 i,根据下面的公式分别计算通信时间 T[i]:



T[i] = 总传输延迟 + 总数据量/数据传输速度(单位:b/s)

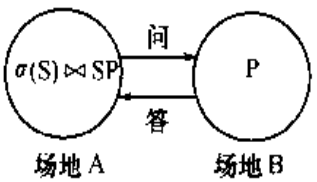
策略 1　把关系 P 传送到场地 A,在 A 地进行查询处理,所以

$$T[1] = 1 + 10^5 \times 100/10^4 = 10^3 \text{ 秒}(16.7 \text{ 分})$$



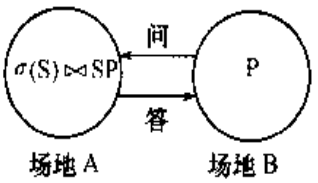
策略 2　把关系 S、SP 传到场地 B,在 B 地执行查询处理,所以

$$T[2] = 2 + (10^4 + 10^6) \times 100/10^4 \approx 10\,100 \text{ 秒}(2.8 \text{ 小时})$$



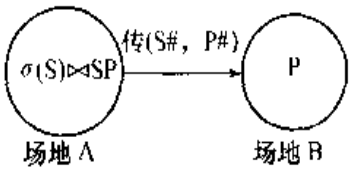
策略 3　在场地 A 连接关系 S 和 SP,选出城市为北京的元组(10^5 个),然后对这些元组中的每个元组的 Pno,询问场地 B,看此零件是否红色。所以共问答 10^5 次,由于不是传送数据,只是消息的问答,所以

$$T[3] = 2 \times 10^5 \text{ s}(2.3 \text{ 天})$$



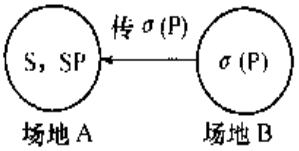
策略 4　在场地 B 选出红色零件的元组(10 个),然后对每一个元组逐一检查场地 A,看北京供应商的装运单中是否有这个零件装运单(若有则选出 S#)每做这样一次检查包括 2 次消息,共问一答 10 次,所以

$$T[4] = 2 \times 10 = 20 \text{ 秒}$$



策略 5　在场地 A 选出北京的供应商的装运单把结果送到场地 B，在场地 B 完成最后处理，所以

$$T[5] = 1 + (10^5 \times 100)/10^4 \approx 1\,000 \text{ 秒}(16.7 \text{ 分})$$



策略 6　在场地 B 的关系 P 中选出红色的元组(10 个)，把结果送到场地 A 完成最终处理。所以

$$T[6] = 1 + (10 \times 100)/10^4 \approx 1 \text{ 秒}$$

表 14.1 综合了上述结果。查询处理时间还应包括在某一场地上的处理时间。这和集中数据库系统中查询处理时间计算是一样的，主要包括 I/O 时间和 CPU 时间。由于在分布式数据库中通信时间是最主要的开销，这里只考虑查询处理中的通信时间。在这个例子中，忽略了发出查询的地点，即最后结果传送到哪个站的问题。在实际系统中还应加上传送结果的开销。

表 14.1　分布环境下查询策略实例比较

策　略	通信时间	方　法
1	16.7 分	把 P 传到场地 A
2	2.8 小时	把 S,SP 传到场地 B
3	2.3 天	对每一北京的装运单，检查相应零件是否红色
4	20 秒	对每一红色零件，检查北京供应商中是否有人供应
5	16.7 分	把北京供应商的装运单传送到场地 B
6	1 秒	把红色零件传送到场地 A

这个例子说明了：

(1) 不同的存取策略通信时间相差很大，达多个数量级！因此必须进行优化。

(2) 在有些策略中数据传输速度和传输延迟都要考虑；而在有些策略中(如例子中策略 3、策略 4)主要考虑传输延迟，还有一些策略中(如例子中策略 1、策略 2、策略 5)数据传输量大，主要考虑传输时间。

此外，有些策略还允许在两个场地上并行处理。例如策略 6，可以在场地 A 完成对北京供应商装运单选择、连接的同时，在场地 B 选出红色零件的元组，因此总的查询时间还可能比集中数据库系统中所花费的要少。

14.3.2　查询处理和优化要解决的问题

绝大多数分布式数据库系统都是关系型的，由于关系查询的语义级别较高，为查询优化提供了可能。系统执行查询可以有多种策略，而且彼此之间性能会有很大差别。分布 DBMS 极其复杂，系统开销非常可观，为了提供能够接受的效率，查询优化是十分必要的。

在分布式数据库系统中有三类查询:局部查询、远程查询和全局查询。局部查询和远程查询都只涉及单个结点上的数据(本地的或远程的),所以查询优化采用的技术就是集中式数据库的查询优化技术(代数优化和非代数优化)。全局查询涉及多个结点的数据,因此查询处理和优化要复杂得多,下面要讨论的是全局查询处理和优化涉及的问题,优化的目标及连接查询的优化方法。

为了执行全局查询和确定一个好的查询处理策略,要做许多判断、计算工作,但总体上可分为三类。

1. 查询分解

对于全局查询必须把它们分为若干子查询,每个查询只涉及某一结点的数据,可以由局部 DBMS 处理。在分布式数据库中一个关系可分为若干逻辑片段,这些片段又可以在系统的多个结点上存放。所以,对一个查询中所涉及的关系需要确定一个物理片段,选择不同的物理片段执行查询操作会直接影响查询执行的效率。因此必须选择查询开销最省的那些物理片段。

2. 选择操作执行的次序

主要是确定连接和并操作的次序,其他的操作顺序是不难确定的,例如选择和投影操作总是应尽量提前执行。这个原则和集中数据库中代数优化的策略相同。但是,涉及不同结点上关系的连接和并操作的次序是必须认真考虑的。

3. 选择执行操作的方法

这包括将若干操作组合在对数据库的一次存取中执行;选择可用的存取路径(如索引)以及选择某一种算法等问题。连接(Join)是查询中最费时的操作,因此连接的执行方法是研究的重点。在分布式数据库中提出了半连接(Semi-Join)的方法。

显然,这 3 个问题不是独立的。为了找到一个全局查询的最优分解,确定执行查询的物理片段,必须了解查询中各操作执行的次序,而这又依赖于对每个操作的执行方法。而且,这 3 个问题之间的联系不是单向的或线性的,而是互相制约的。不过,这 3 个问题中关键是第二个问题。因为第一、第三个问题可将"使应用的局部性最大化"和"使应用的并发性最大化"作为准则,而第二个问题则需要大量计算比较,需要关于数据库的环境信息(如关系的大小,连接属性上的值的分布,通信线路的有关指标等)才能解决。

14.3.3 查询优化的目标

无论是在集中式数据库中还是在分布式数据库中一个查询处理策略的选择都是以执行查询的预期代价为依据的。不同的只是构成一个查询代价的主要因素在这两类系统中不完全一样。

在集中式数据库中,查询执行开销主要是

I/O 代价 + CPU 代价

而在分布式数据库中，除上面两种开销外还有数据在网络上的传输代价：

I/O 代价 + CPU 代价 + 通信代价

对于分布式数据库中数据传输代价应区别不同情况考虑。在远程通信网或数据传输率较低的系统中，通信代价可能会比查询执行中的 I/O 及 CPU 开销大得多，因而作为首要的优化目标来考虑。在局域网且传输率高的系统中，通信代价和局部处理的开销差不多，此时就应在优化中将它们平等对待。

但是，在查询优化过程中将通信代价作为一个首要问题单独列出来研究，从以下两点看来似乎更合理：

首先，通信代价很好估计，通常是数据传输量的一个函数，这个特点是 I/O 开销所不具备的。

其次，分布查询可分为两部分：存取策略的分布优化和局部优化，二者可以分别予以解决。其中，局部优化可以采用集中式数据库中的技术，而分布优化是分布系统要考虑的，一般来说比局部优化更重要。

通信代价可以用下面的公式粗略计算：

$$TC(X) = C_0 + X * C_1$$

其中 X 是数据传输量，这里以 b(位)为单位计算；C_0 为两结点之间初始化一次传输所花费的开销，它由通信系统决定，近似为一个常数，单位为 s(秒)；C_1 为单位数据(b)传输的代价，单位为(s/b)。

总之，在分布式数据库中查询优化的首要目标是：使该查询执行时其通信代价最省。

不同结点之间的连接操作和并操作是数据传输的主要原因，因此连接查询的优化在分布优化中是举足轻重的。

14.3.4　连接查询的优化

对连接查询的优化可以采取两种策略，一种是使用半连接来缩减关系(或片段)进而节省传输开销；另一种是不采用半连接，而是用直接连接的优化方案。这里重点讨论半连接的方法。

一、用半连接来优化连接的方法

设 R、S 是两个关系，A、B 分别是 R、S 上的两个属性。

首先引入半连接的记号：$\ltimes$

R、S 在 A、B 上的半连接可表示为：

$$R \underset{A=B}{\ltimes} S = R \underset{A=B}{\bowtie} (\pi_B(S))$$

一个连接运算可以用半连接作为中间步骤来实现：

$$R \underset{A=B}{\bowtie} S = (R \underset{A=B}{\ltimes} S) \underset{A=B}{\bowtie} S$$

具体操作过程和执行代价如下：

设关系 R、S 分别存放在结点 r、s 上(如图 14.7 所示)。

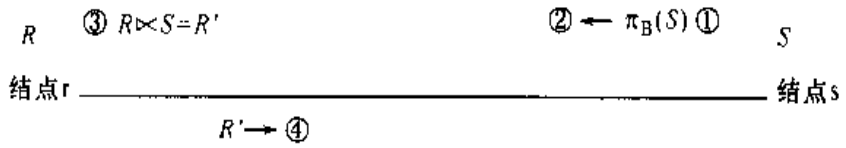


图 14.7

① 在结点 s 作关系 S 在属性 B 上的投影 $\pi_B(S)$

② 把 $\pi_B(S)$ 送到结点 r,代价为

$$C_0 + C_1 \times \text{size}(B) \times \text{val}(B[S])$$

其中,size(B)表示属性 B 的长度,val(B[S])表示关系 S 中属性 B 上不同值的个数。

③ 在结点 r 计算半连接,设结果为 R',即

$$R' = R \underset{A=B}{\ltimes} S$$

④ 把 R' 从结点 r 送到结点 s,代价为:

$$C_0 + C_1 \times \text{size}(R) \times \text{card}(R')$$

card(R')表示 R' 的元组数。

⑤ 在结点 s 执行连接操作:

$$R' \underset{A=B}{\bowtie} S$$

在上面的操作过程中,忽略了局部场地上执行操作的代价,仅考虑了通信代价。

直观地看,使用半连接可以缩减关系 R 的大小使得从结点 r 传送的关系由 R 缩减为 R'(一般 $\text{card}(R) > > \text{card}(R')$),从而减少了通信代价,从中获得好处。

但是,由于执行半连接本身要传输数据($\pi_B(S)$),需要通信代价,因而究竟是否采用半连接还要计算和比较其好处和开销之后再作决定。

采用半连接方案的总代价为

$$C_{SJ} = 2C_0 + C_1(\text{size}(B) \times \text{val}(B[S]) + \text{size}(R) \times \text{card}(R'))$$

不用半连接,直接把 R 送到 S 执行连接的代价为

$$C_{JN} = C_0 + C_1 \times \text{size}(R) \times \text{card}(R)$$

只有当 $C_{SJ} < C_{JN}$ 时,采用半连接方案才是合适的。

上面分析的是简单的连接查询,对于复杂的连接查询,即包括多个关系的连接,则可能存在许多可应用的半连接方案,而其中总有一个方案的效果最佳,因此优化的第一步是从所有半连接方案中找出一个最优的。第二步是与不用半连接的方案进行比较,选出较优者。

由美国计算机公司(Computer Corporation of America)研制的 SDD-1 分布式数据库系统就是采用的半连接的优化方案。

二、用直接连接的方案

上面已经讲过,用半连接方案并不总是有利可图的,因此有些系统如 R*(美国 IBM San

Jose 研究室研制的系统 R 的分布版本)就采用了直接做连接的方案。直接做连接也有不同的操作方法可选用。R* 给出了以下几种可选的方法,它们的算法思想与集中库中的相同:

- 嵌套循环法;
- 排序 - 合并方法。

R* 支持两种传输方式:

- 整体传输:把整个关系传至指定结点再作处理;
- 按需传输:逐个取元组,只传所需要的元组。

以上两种连接算法和传输方式的组合构成了 R* 优化过程中的候选方案,经过代价计算选出其中最省的方案作为最终的查询计划。

14.4 分布事务管理

一个事务的执行必须保持其原子性,即它所包含的所有更新操作要么全部都做,要么都不做。

在分布式数据库系统中,一个全局事务会涉及多个场地上的数据更新。因此,事务是分布执行的。可以把一个事务看成由不同场地上的若干子事务组成。分布事务的原子性就应该是:组成该事务的所有子事务要么一致地全部提交,要么一致地全部回滚。

在多用户系统中,还必须保证分布事务的可串行性。因此,分布事务管理主要包括两个方面:事务的恢复和并发控制。这一节简要地讨论这两个问题。

14.4.1 分布事务的恢复

数据库在运行过程中故障和错误总是难免的。不同的故障会造成数据库不同程度的损害,轻则事务不能正常运行,重则造成数据库数据不一致性。更严重的情况,会使部分或整个数据库遭到破坏。我们已在第七章中讨论了集中式数据库系统中的故障类型和恢复技术,在集中式数据库系统中利用日志文件(必要时加上后备副本)总可以把数据库恢复到某种正确的状态。

分布式数据库系统中,各个场地除了可能发生如同集中式数据库的那些故障外,还会出现通信网络中通信信息丢失、长时间延迟、网络线路中断等事故。因此,情况比集中式数据库更复杂,相应的恢复过程也就更复杂些。

为了执行分布事务,通常在每个场地都有一个局部事务管理器,用来管理局部子事务的执行,保证子事务的完整性。同时这些局部管理器之间还必须相互协调,保证所有场地对它们所处理的子事务采取同样的策略:要么都提交,要么都回滚。为了保证这一策略,最常用的技术是两段提交协议(2-Phase-Commitment Protocol,2PC)。

两段提交协议把一个分布事务的事务管理分为两类:一个是协调者,所有其他的是参与

者。协调者负责作出该事务是提交还是撤销(Abort)的最后决定。所有参与者负责管理相应子事务的执行及在各自局部数据库上执行写操作。

两段提交协议的内容是:

(1) 第一阶段:协调者向所有参与者发出"准备提交"信息。如果某个参与者准备提交,就回答"就绪"(Ready)信息,否则回答"撤销"信息。参与者在回答前,应把有关信息写入自己的日志中。协调者在发出准备提交信息前也要把有关信息写入自己的日志中。

如果在规定时间内协调者收到了所有参与者"就绪"的信息,则将作出提交的决定,否则将作出撤销的决定。

(2) 第二阶段:协调者将有关决定的信息先写入日志,然后把这个决定发送给的所有的参与者。所有参与者收到命令之后首先往日志中写入"收到提交(或撤销)"决定的信息,并向协调者发送"应答(ACK)"消息,最后执行有关决定。协调者收到所有参与者的应答消息后,一个事务的执行到此结束,有关日志信息可以脱机保存。

采用两段提交协议后,当系统发生故障时,各场地利用各自有关的日志信息便可执行恢复操作,恢复操作的执行类似集中式数据库。

可以看出,在两阶段提交协议中,各结点采取的是完全同步的方法来保证数据库的一致性,称为紧密一致性(Tight Consistency)。

紧密一致性能绝对保证任何时刻整个分布式数据库系统的数据一致性和全局事务的原子性,但从实际应用的角度来说,它的缺陷也是非常明显的。

(1) 全局事务可靠性低,任何一个参与结点的失败或网络的中断都将导致整个事务的回滚;

(2) 系统效率低下,由于采用完全同步的方法来保证数据库的一致性,系统的性能将取决于系统中最慢的结点。

为了解决紧密一致性的缺陷,相应地提出了松散一致性(Loose Consistency)的概念。数据各副本的修改是异步的,也就是说各副本将不保证任何时刻数据库绝对的一致性;各副本同步的延迟时间可长可短,视系统的具体情况而定。与紧密一致性相比,松散一致性比较灵活,系统的可用性大为提高,适用于对数据一致性要求不是很高的应用,但对于银行转账这样的关键性应用显然是不适合的。

14.4.2 并发控制

集中式数据库系统中并发控制一般采用封锁技术。

锁可分为不同类型,常用的是共享锁(SLock)、排它锁(XLock)。封锁的对象可以是表一级的或记录一级的。

为了保证并发事务的可串行性,必须做到:

- 遵守锁的相容性规则;

• 遵守两段锁(2-Phase Lock)协议,即如果一个事务已释放了一个锁,那么它就不能再请求新的封锁。

在分布式数据库系统中并发控制也可采用封锁技术,不过与集中式数据库系统相比,由于下面两个原因,使并发控制更为复杂。

(1) 分布式数据库系统支持多副本的特点;

(2) 由于事务的分布执行,封锁的方法会引起全局死锁。

因此分布式数据库系统中基于封锁技术的并发控制必须解决这两个问题。

分布式数据库系统支持多副本的特点给并发控制带来了困难。例如事务 T_1 在场地 1 获得了对数据 d 的 X 封锁,而事务 T_2 在场地 2 对数据 d 的副本申请 X 封锁,场地 2 的事务管理器并不知道场地 1 上的 d 已被 T_1 加了 X 封锁,因此事务 T_2 仍获得对 d 的 X 封锁,这样,封锁机制就失去了原来的功能而完全无用了。

T_1在场地 1	T_2在场地 2
$Xlock(d_{site1})$	$Xlock(d_{site2})$

为了避免这个问题,分布事务管理就要把"事务 T_1 对 d 的 X 封锁"这件事让 d 副本所在场地上的事务管理器都知道,一个简单的方法是向这些场地的事务管理器发出局部封锁请求,这个办法是有效的。但封锁的冗余度很大,局部封锁的数目和副本数相同。为了减少系统开销,处理多副本的封锁可采取如下几种方法:

(1) 对写操作,要申请对所有副本的 X 锁。对于读操作,只要申请对某个副本的 S 锁。

(2) 无论是写操作还是读操作都要对多数(大于半数)副本申请 X 锁或 S 锁。

(3) 规定某个场地上的副本为主副本,所有的读写操作均申请对主副本的封锁。

这 3 个方法均可有效地发现冲突,协调并发事务的执行。

基于封锁的并发控制方法在分布环境下还必须解决全局死锁的问题,所谓全局死锁即包括两个以上场地上的死锁。

和集中式数据库系统相似,通常采用分布等待图的方法来检测死锁。不同的是,分布情况下死锁检测涉及多个场地,多个局部数据库,需要较多的通信和验证,开销较大。

除了用死锁检测及解除方式来解决死锁问题外,还可以采用死锁预防方法,即不让死锁发生。典型的一种解决方法是对事务按某一标准进行排序,只允许它们沿着这一次序单向等待,这就不会发生死锁了。

在分布式数据库中还研究了基于时标(Time Stamp,亦称为时间戳)的方法和乐观方法等并发控制技术,但实际系统则大都是基于封锁的方法。

14.5 小 结

由于应用的驱动,数据库技术和网络技术的进展,分布式数据库技术得到了长足的

发展。

这一章讨论了分布式数据库系统的基本概念、这一领域内的某些主要问题以及解决这些问题所采用的基本技术和基本方法。

分布式数据库系统以集中式数据库系统技术为基础，但不是简单地把集中式数据库连网就能实现的。分布式数据库系统具有自己的性质和特征。集中式数据库的许多概念和技术，如数据独立性、数据共享和减少冗余度、并发控制、完整性、安全性和恢复等在分布式数据库系统中都有了不同的更加丰富的内容。

习　题

1. 什么样的数据库系统是分布式数据库系统？图 14.1 的系统配置在什么情况下只能算分散的数据库系统？在什么条件下才是分布式数据库系统？

2. 分布式数据库系统有什么特点？

3. 试述研制分布式数据库系统的目的和动机。

4. 试述分布式数据库系统的模式结构。

5. 什么是数据分片？有几种分片方式？数据分片的目的是什么？有什么优点？

6. 试述分布透明性的内容。

7. 什么是同构型 D-DBMS？什么是异构型 D-DBMS？

8. 设在 14.2.3 节的分布式数据库系统例子中，还有全局关系 SC(SNO,CNO,G)，它具有两个导出分片 SC_A,SC_B，分别存储理学院和文学院学生的选课记录。SC_A 存放在场地 4，SC_B 存放在场地 5。今有一个稍复杂的查询，从终端输入一个课程号，查找选修该课程的学生学号和姓名，并把它们显示在屏幕上。请写出具有不同层次分布透明性(类比例子中的三种情况)的应用程序。不必给出细节，只需写出算法思想。

9. 对 14.3.1 节的例子中介绍的六种策略改用下列估算值后分别计算通信时间：

红色零件数 = 1 000，

北京供应商的装运单 = 10 000。

10. 设关系 R,S 存储在不同的场地，已知：

$\mathrm{card}(R) = 100, \mathrm{size}(R) = 50$

$\mathrm{card}(S) = 50, \mathrm{size}(S) = 5$

今要计算 $R \underset{A=B}{\bowtie} S$，其中 A,B 分别为 R,S 上的属性，并且

$\mathrm{val}(A[R]) = 50, \mathrm{size}(A) = 3$

$\mathrm{val}(B[S]) = 50, \mathrm{size}(B) = 3$

若 $R' = R \underset{A=B}{\ltimes} S, \mathrm{card}(R') = 0.2 * \mathrm{card}(R)$

$S' = S \ltimes R, \mathrm{card}(S') = 0.8 * \mathrm{card}(S)$

请比较采用以下不同策略时的通信代价：

(a) 在 R 所在场地执行连接操作，利用半连接方法；

(b) 在 S 所在场地执行连接操作，利用半连接方法；

(c) 在 R 所在场地执行连接操作，不用半连接方法；

(d) 在 S 所在场地执行连接操作,不用半连接方法;

哪种情况代价最省(不考虑把结果传送到某一场地的代价)?

11. 试述下列概念:两段提交协议(2PC);分布事务的原子性;全局死锁。

12. 在分布式数据库系统中,对多副本的封锁有几种解决方法?

本章参考文献

1 Rothnie J B, Goodman N, A Survey of Research and Development in Distributed Database Management. in Proceedings of LDB, 1977

(本文综述了分布式数据库系统早期的研究结果。)

2 Ceri S et al. Distributed Databases: Principles and Systems. McGraw Hill, NY, 1984

(本书介绍了分布式数据库系统的基本概念、原理和系统。)

3 周龙骧等. 分布式数据库管理系统实现技术. 数据库技术丛书之一, 北京:科学出版社, 1998

4 郑振楣, 于戈, 郭敏. 分布式数据库. 数据库技术丛书之一, 北京:科学出版社, 1998

5 Tamer Ozsu M, Valduriez P. Principles of Distributed Database Systems. Prentice-Hall International Inc. 1991

6 石树刚, 郑振楣, 阮春等. 分布式数据库系统的实用化. 微型计算机, 1989

(本期为分布式数据库的专刊,刊登了十余篇论文(如文献[6],[7],[8],[9],[10],[11]),反映了国内在这一领域中当时的研究的情况和部分成果。)

7 李曦, 王珊, 萨师煊. 一个基于大型机远程通信的分布式数据库查询系统 DQS/SEIS. 微型计算机, 1989

8 杜小勇, 王珊, 萨师煊. 分布式数据库 DQS/SEIS 设计实现中若干问题的探讨. 微型计算机, 1989

9 张霞, 郑怀远. 面向客体和语义关联的异构分布式数据库系统 OHDDB-DGI. 微型计算机, 1989

10 王能斌, 徐立臻, 程琪. SUNDDB:一个分布式数据库管理系统. 微型计算机, 1989

11 周晓方, 金志权, 柳诚飞. LSZTM:一个分布式数据库系统的管理. 微型计算机, 1989

12 王珊等. The Global Conceptual Model Design in DQS/SEIS. 第二届泛太平洋计算机国际会议 PPCC-3 论文集. 1989

(本文讨论分布式数据库系统全局概念模式的设计,以中国人民大学为国家经济信息系统研制的分布查询系统 DQS/SEIS 为实际背景。)

13 周龙骧, 顾君忠. POREL:一个分布式关系型数据库系统. 计算机科学, 1982

14 Ceri S, Navathe B, Wiederhold G. Distribution Design of Logical Database Schemas. IEEE Transactions on Software Engineering, Vol. SE-9, No. 4, 1983

(本文讨论了分布式数据库系统的设计问题。)

15 Epstein R, Stonebraker M R. Analysis of Distributed Database Processing Strategies. in Proceedings of VLDB, 1980

(本文讨论了分布式数据库中的查询处理策略。)

16 Apers P M G. Yao S B. Optimization Algorithms for Distributed Queries, IEEE Transactions on Software Engineering, Volume SE-9, Number 1, 1983

(本文讨论了分布查询处理的优化算法。)

17 Daniels D, Selinger P G, et al. An Introduction to Distributed Query Compilation in R*, in Distributed Data

Bases. Proceedings of International Symposium on Distributed Databases, 1982

（本文全面介绍了分布式数据库系统 R*，介绍 R* 的论文很多，这只是其中之一。）

18 Berntstein P A, Chiu D W. Using Semijoins to Solve Relational Queries. Journal of the ACM, Volume 28, Number 1, 1981

（本文讨论了分布查询处理的半连接优化方法。）

19 Kambayashi Y, Yoshikawa M, Yajima S. Query Processing for Distributed Databases Using Generalized Semi-joins. Proceedings of the ACM SIGMOD, 1982

（本文讨论了分布查询处理的半连接优化方法。）

20 Traiger I L, et al. Transactions and Consistency in Distributed Database Management Systems. ACM TODS, Volume 7, Number 3, 1982

（本文讨论了分布式数据库系统中事务及其一致性的概念。）

21 Bernstein P A, et al. Concurrency Control in a System for Distributed Databases (SDD - 1). ACM TODS, Volume 5, Number 1, 1980

（本文讨论了分布式数据库的并发控制问题。）

22 Bernstein P A, Goodman N. Timestamp-based Algorithms for Concurrency Control in Distributed Database Systems. in Proceedings of VLDB, 1980

（本文讨论了分布式数据库系统中基于时间戳的并发控制方法。）

23 Kohler W H. A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems. ACM Computing Surveys, Volume 13, Number 2, 1981

（本文讨论了分布式数据库系统中的同步和数据库恢复问题。）

24 Lampson B, Sturgis H. Crash Recovery in a Distributed Data Storage System. Technical Report, Computer Science Laboratory, Xerox, Palo Alto Research Center, Palo Alto, CA, 1976

（本文提出了分布式数据库系统中事务的两段式提交协议。）

第十五章　对象关系数据库系统

本章介绍对象关系数据库系统(Object Relational DataBase System,**ORDBS**)。ORDBS是面向对象数据模型(Object Oriented Data Model,以下简称**OO**模型)和关系数据模型相结合的产物。

数据库技术在商业领域的巨大成功刺激了其他领域对数据库技术需求的迅速增长。这些新的领域既为数据库应用开辟了新天地,又提出了新的数据管理需求,给数据库技术提出了挑战。新应用的挑战主要来自多种数据类型的应用领域,例如,计算机辅助排版系统中的大文本,天气预报中的图像,工程设计中复杂的图形数据,股票市场交易历史中的时间序列数据,空间和地理数据,等等。由于传统数据库系统的设计目标源于商业事务处理,它们难以适应和满足新的数据库应用需求。例如,RDBMS只支持简单有限的数据类型,不支持用户自定义的数据类型;不支持用户自定义的运算和函数;不能清晰地表示复杂对象;没有全系统唯一的标识符等。这些是关系模型所不支持的。为此人们研究了非一范式的关系模型、语义数据模型、面向对象数据模型(Object Oriented Data Model)等。其中以面向对象数据模型最有影响。从20世纪80年代起,国内外学术界、工业界努力探索数据库技术与对象技术的结合,沿着三条路线展开了OO数据模型和面向对象数据库系统的研究:

一条是以面向对象的程序设计语言为基础,研究持久的程序设计语言,支持OO模型;

一条是建立新的面向对象数据库系统OODBS,支持OO数据模型;

一条是以关系数据库和SQL为基础,把面向对象技术融入数据库系统的ORDBS。

回顾上述3个方向所做的研究及开发工作,可以看到它们的发展是不平衡的。其中ORDBS方向的成果最为卓著。它在传统关系数据库的基础上吸收了OO模型的主要思想,同时又保持了关系数据库系统的优点,成功开发了诸如Postgres,Illustra等原型系统。近年来,各大RDBMS厂商也已推出了其产品的对象-关系版本,从面满足了许多新的数据库应用需求。

面向对象数据模型是继关系数据模型之后最重要的数据模型。本章15.1节首先简单介绍OO模型的基本概念,然后下面各节讲解ORDBS。

15.1 面向对象数据模型

面向对象数据库系统支持**OO模型**。

也就是说，一个**面向对象数据库系统**是一个持久的、可共享的对象库的存储和管理者；而一个对象库是由一个OO模型所定义的对象的集合体。

15.1.1 OO模型的核心概念

一个OO模型是用面向对象观点来描述现实世界实体(对象)的逻辑组织、对象间限制、联系等的模型。一系列面向对象核心概念构成了OO模型的基础。概括起来，OO模型的核心概念有如下几个。

1. 对象(Object)

对象是由一组数据结构和在这组数据结构上的操作的程序代码封装起来的基本单位。对象通常与实际领域的实体对应。一个**对象**包括以下几个部分。

- 属性(Attribute)集合：所有属性合起来构成了对象数据的数据结构(有的书上称为变量集合)。属性描述对象的状态、组成和特性。对象的某一属性可以是单值的或值的集合，也可以是一个对象，即对象可以嵌套。这种嵌套可以继续，从而组成各种复杂对象。
- 方法(Method)集合：方法描述了对象的行为特性。方法的定义包括两部分，一是方法的接口，二是方法的实现。方法的接口用以说明方法的名称、参数和结果返回值的类型，也称之为调用说明。方法的实现是一段程序编码，用以实现方法的功能，即对象操作的算法。

2. 对象标识OID(Object IDentifier)

面向对象数据库中的每个对象都有一个唯一的不变的**标识**称为**对象标识**(OID)。**对象**标识具有**永久持久性**，即一个对象一经产生系统就会赋于一个在全系统中唯一的对象标识符，直到它被删除。OID是由系统统一分配的，唯一的，用户不能对OID进行修改。因此，OID与关系数据库中码(Key)的概念和某些关系系统中支持的记录标识(RID)、元组标识(TID)是有本质区别的。OID是独立于值的、系统全局**唯一的**。

3. 封装(Encapsulation)

每一个对象是其状态与行为的**封装**，其中状态是该对象一系列属性值的集合，而行为是在对象状态上操作的集合，操作也称为方法。

OO模型的一个关键概念就是封装。每一个对象是其状态与行为的封装。

封装是对象的外部界面与内部实现之间实行清晰隔离的一种抽象，外部与对象的通信只能通过消息，这是OO模型的主要特征之一。

但是，对象封装之后查询属性值必须通过调用方法，不能像关系数据库系统那样(用

SQL)进行即席的(随机的)、按内容的查询,这就不够方便灵活,失去了关系数据库的重要优点,因此在 OODB 中必须在对象封装方面作必要的修改或妥协。

4. 类(Class)

共享同样属性和方法集的所有对象构成了一个**对象类**(简称**类**),一个对象是某一类的一个**实例**(instance)。例如,学生是一个类,李勇、刘晨、王敏、张立等是学生类中的对象。在数据库系统中,一再要大家注意区分"型"和"值"的概念。在 OODB 中,类是"型",对象是某一类的一个"值"。

类属性的定义域可以是任何类,即可以是基本类,如整数、字符串、布尔型,也可以是包含属性和方法的一般类。特别地,一个类的某一属性的定义也可是这个类自身。

可以看到,类的概念类似关系模式,类的属性类似关系模式中的属性,对象类似元组的概念,类的一个实例对象类似关系中的一个元组。

可以把类本身也看作一个对象,称为**类对象**(Class Object)。

面向对象数据库模式是类的集合。

15.1.2　类层次(结构)

在一个面向对象数据库模式中,会出现多个相似但又有所不同的类。例如,教职员工和学生两个类都有身份证号,姓名、年龄、性别、住址等属性,也有一些相同的方法。当然,教职员工对象中有一些特殊的属性和方法,如工龄、工资、办公室电话号码、家庭成员数。教职员工和学生的公共属性和方法部分可以统一定义,各自的特殊属性和方法部分可以分别定义。为此,面向对象的数据模型提供了一种**类层次结构**。

例如,可以定义一个类"人"。人的属性和方法是教职员工和学生的公共属性和公共方法。教职员工类和学生类定义为人的子类。教职员工类只包含教职员工的特殊属性和特殊方法。学生类也只包含学生的特殊属性和特殊方法,如图 15.1 所示。

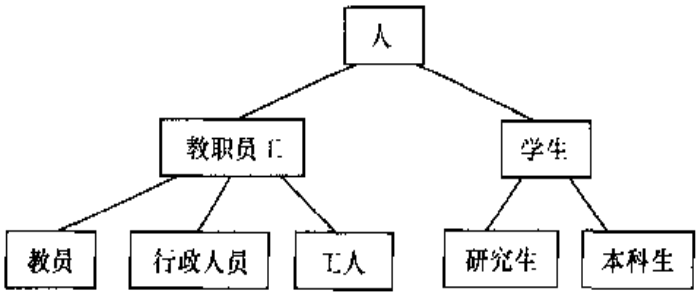


图 15.1　学校数据库的类层次结构图

为了叙述简单,没有给出这些类的方法。在这个类层次中教职员工和学生是人的子类,教员、行政人员和工人是教职员工的子类。教员、行政人员、工人中实际只有它本身的特殊属性和方法,同时它们又继承教职员工类和人的所有属性和方法。因此,逻辑上它们具有人、教职员工和本身的所有属性和方法。同样,本科生和研究生是学生的子类。

超类/子类之间的关系体现了"ISA"的语义,例如,图 15.1 中教员"ISA"教职员工,教职员工"ISA"人。因此,超类是子类的**抽象**(Generalization)或**普通化**,子类是超类的**特殊化**

(Specialization)或**具体化**。

类层次可以动态扩展,一个新的子类能从一个或多个已有类导出。根据一个类能否继承多个超类的特性将继承分成了单继承和多重继承。

15.1.3 继承

在OO模型中常用的有两种继承,**单继承**与**多重继承**。若一个子类只能继承一个超类的特性(包括属性和方法),这种继承称为单继承;若一个子类能继承多个超类的特性,这种继承称为多重继承。例如,在学校中实际上还有"在职研究生",他们既是教员又是学生,在职研究生继承了教职员工和学生两个超类的所有属性和方法,如图15.2所示。

单继承的层次结构图是一棵树(如图15.1),多继承的层次结构图是一个带根的有向无回路图(如图15.2)。

继承性有两个优点,第一,它是建模的有力工具,提供了对现实世界简明而精确的描述。第二,它提供了信息重用机制。由于子类可以继承超类的特性,这就可以避免许多重复定义。当然子类除了继承超类的特性外还要定义自己特殊的属性和方法。在定义这些特殊属性和方法时可能与继承下来的超类的属性和方法发生冲突,例如在教职员工类中已经定义了一个操作"打印",在教员子类又要定义一个操作"打印",这就产生了同名冲突。这类冲突可能发生在子类与超类之间,也可能发生在子类的多个直接超类之间。这类冲突通常由系统解决,在不同的系统中使用不同的冲突解决方法,便产生了不同的继承性语义。

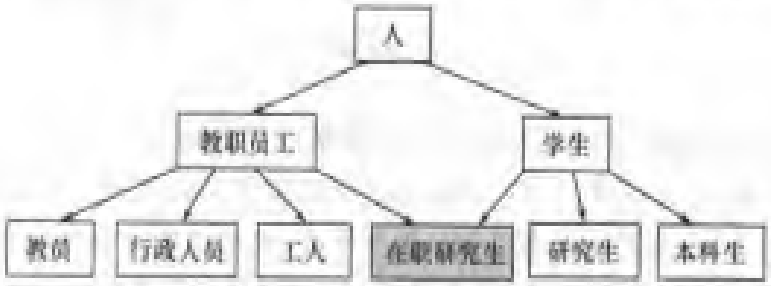


图15.2 具有多继承的类层次结构图

例如,对于子类与超类之间的同名冲突,一般是以子类定义的为准,即子类的定义取代或替换由超类继承而来的定义。对于子类的多个直接与超类之间的同名冲突,有的系统是在子类中规定超类的优先次序,首先继承优先级最高的超类的定义;有的系统则指定继承其中某一个超类的定义。

子类对父类即有继承又有发展,继承的部分就是重用的成分。由封装和继承还导出面向对象的其他优良特性,如多态性、动态联编等。

15.1.4 对象的嵌套

前面已经讲到,在一个面向对象数据库模式中,对象的某一属性可以是单值的或值的集

合。进一步地，一个对象的属性也可以是一个对象，这样对象之间产生一个**嵌套层次结构**。对象嵌套概念是面向对象数据库系统中又一个重要概念。

设 Obj_1 和 Obj_2 是两个对象。如果 Obj_2 是 Obj_1 的某个属性的值，称 Obj_2 属于 Obj_1，或 Obj_1 包含 Obj_2。一般的，如果对象 Obj_1 包含对象 Obj_2，则称 Obj_1 为**复杂对象**或**复合对象**。

Obj_2 是 Obj_1 的组成成分，也可称是 Obj_1 **子对象**。Obj_2 还可以包含对象 Obj_3，这样 Obj_2 也是复杂对象，从而形成一个嵌套层次结构。

[例 1] 每辆汽车包括：汽车型号、汽车名称、发动机、车体、车轮、内部设备等属性。其中汽车型号和汽车名称的数据类型是字符串，发动机不是一个标准数据类型，而是一个对象，包括：发动机型号、马力等属性；车体也是一个对象，包括：钢板厚度、钢板型号、车体形状等属性；内部设备也是一个对象，包括：车座、音响设备、安全设备等属性；音响设备也可以是一个对象，包括：VCD、喇叭等属性，如图 15.3 所示。

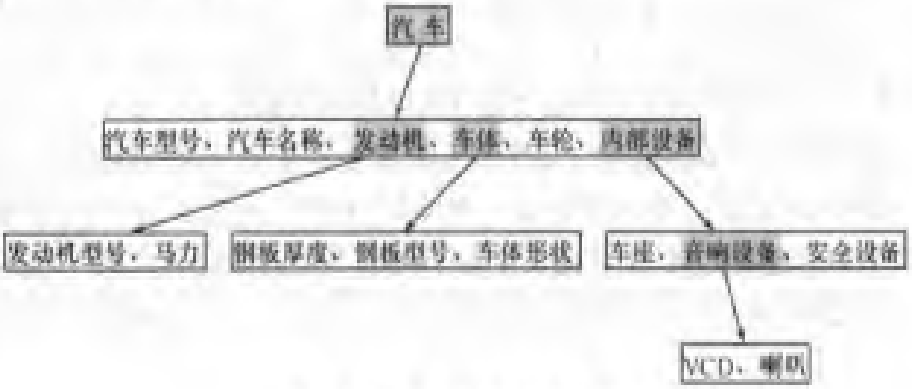


图 15.3 汽车的嵌套层次图

对象嵌套层次结构和类层次结构形成了对象横向和纵向的复杂结构。这是说，不仅各种类之间具有层次结构，而且某一个类内部也具有嵌套层次结构，它不再像一个关系模式那样是平面结构的了，一个类的属性可以是一个基本类，也可以是一个一般类。

作为本节小结，表 15.1 给出了关系数据模型与 OO 模型主要特性的比较。

表 15.1 关系数据模型与 OO 模型的比较

内容	关系数据模型	面向对象数据模型
基本数据结构	二维表	类
数据标识符	码	OID
静态性质	属性	属性
动态行为	关系操作	方法
抽象数据类型	无	有
封装性	无	有
数据间关系	主外码联系，数据依赖	继承、组合
模式演化能力	弱	强

15.2 对象－关系数据库

ORDBS 是关系数据库系统与面向对象数据模型的结合。它保持了关系数据库系统的非过程化数据存取方式和数据独立性，继承了关系数据库系统已有的技术，支持原有的数据管理，又能支持 OO 模型和对象管理。

SQL3 是 1999 年发布的 SQL 标准，也称为 SQL99。其显著的特点之一是提供了面向对象的扩展，增加了 SQL/Object Language Binding。SQL3 的扩展使我们可以同时处理关系模型中的表和对象模型中的类与对象。SQL3 对 ORDBMS 标准的制定滞后于实际系统的实现。所以各个 ORDBMS 产品在支持对象模型方面虽然思想一致，但是所采用的术语、语言语法、扩展的功能都不尽相同。下面的讲解以 SQL3 为主要参考，适当加上某些产品的功能和特点。读者在使用具体产品时要以各个产品的手册为准。

15.2.1 对象关系数据库系统中扩展的关系数据类型

SQL3 对关系数据库的数据类型进行了扩展，克服原来关系数据库数据类型单一的缺点。扩展的类型包括：LOB、BOOLEAN、集合类型 ARRAY、用户定义的 DISTINCT 类型等。SQL3 最重要的扩展是面向对象的数据类型，包括行类型 ROW TYPE 和抽象数据类型（Abstract Data Type）。本小节介绍对关系数据类型的扩展，后面章节介绍对面向对象的数据类型的扩展及其定义语言。

1. 大对象 LOB（Large OBject）类型

LOB 可存储多达十亿字节的串。LOB 又分为二进制大对象BLOB（Binary Large OBject）和字符串大对象 CLOB（Character Large OBject）两种。BLOB 用于存储音频、图像数据。CLOB 用于存储长字符串数据。

LOB 类型数据是直接存储在数据库中，由 DBMS 维护，不是存在外部文件中。LOB 类型可以像其他类型的数据一样被查询、提取、插入和修改。但是使用时要注意，必须为 LOB 提供足够大的缓冲区，而且还有很多限制。

应用程序在操作 LOB 类型数据时用 LOB 定位器（LOB locator）来提取 LOB 数据。应用程序通常并不传输整个 LOB 类型的值，而是通过 LOB 定位器访问大对象值。

2. BOOLEAN 类型

BOOLEAN 是布尔类型，它支持 3 个真值：true、false 和 unknown。除了传统的 NOT、AND 和 OR 布尔操作符外 SQL3 还增加了两个新的操作符：EVERY（而不是 ALL）和 ANY。这两种操作的参数都是 BOOLEAN 类型，该 BOOLEAN 值通常是由一个表达式得到的。

例如，WHERE EVERY（QTY >200）或 WHERE ANY（QTY >200）。

如果 QTY 列为空值，两个操作符都返回 unknown；

如果 QTY 列为非空,那么当该列的每一个值都使(QTY > 200)为 true 时,EVERY 返回 true,否则为 false;

当该列的每一个值都使(QTY > 200)为 false 时,ANY 返回 false,否则为 true。(即只要该列中的某一个值使(QTY > 200)为 true 时,ANY 就返回 true)

3. 集合类型(Collection Type)ARRAY

相同类型元素的有序集合称为数组 ARRAY,这是 SQL3 新增的集合类型,它允许在数据库的一列中存储数组。

[例 2]

```
CREATE TABLE SALES
(
 ITEM_NO CHAR(20),                    /* 商品号 */
 QTY INTEGER ARRAY[12],               /* 整数数组,存放 12 个月的销售额 */
 PRIMARY KEY(ITEM_NO)
);
```

SQL3 的数组只能是一维的,而且数组中的元素不能再是数组。

下面是向 SALES 表插入一个元组:

```
INSERT INTO SALES(ITEM_NO,QTY)VALUES
('T-shirt2000',ARRAY[200,150,200,100,50,70,80,200,10,20,100,200]);
```

下面是查询语句,查找三月份销售额大于 100 的商品号:

```
SELECT ITEM_NO
FROM SALES                            /* 从 SALES 表中选出满足下面条件的商品号 */
WHERE QTY[3] > 100;                   /* QTY 数组第三个值大于 100 */
```

4. DISTINCT 类型

SQL3 新加了一种 DISTINCT 类型。

在应用中,不同的字段经常定义成同一种数据类型。例如,职工的智商字段(IQ)和鞋号字段(SHOE_SIZE)都定义成 INTEGER 类型。表达式:WHERE SHOE_SIZE > IQ 语法是对的,因为它们是同一种数据类型,可以相互比较。但是,这显然不符合实际应用的语义。

这时可将 SHOE_SIZE 和 IQ 字段定义成 DISTINCT 类型。语法如下:

```
CREAT TYPE < type name >                          /* 用户定义数据类型的名称 */
AS < built in scalar type name > FINAL            /* 只限于 SQL 内置的那些数据类型 */
 [ < cast option > ]                              /* cast 选项 */
 [ < method specification commalist > ];          /* 定义类型上的操作或方法 */
```

本例中两字段类型可分别定义如下:

```
CREAT TYPE SHOE_SIZE_TYPE AS INTEGER FINAL;
        /* 用户定义 SHOE_SIZE_TYPE 数据类型,它是整型的 */
CREAT TYPE IQ_TYPE AS INTEGER FINAL;
        /* 用户定义 IQ _TYPE 数据类型,它是整型的 */
```

这样就定义了 SHOE_SIZE_TYPE 和 IQ _TYPE 两种数据类型,它们成为两种不同的数据类型。此时 SHOE_SIZE 只能与同类型数据值相比,而不能与其他任何数据类型相比,包括 INTEGER 类型。IQ 也一样。因此,表达式:

WHERE SHOE_SIZE > IQ 就是非法的了。

然而,如果在定义类型时设置了选项 < cast option > ,下面用法也是合法的:

```
WHERE MY_SHOE_SIZE > CAST (MY_IQ AS SHOE SIZE)
```

15.2.2 对象关系数据库系统中扩展的对象类型及其定义

为了支持 OO 数据模型,SQL3 扩展了面向对象的类型系统。在 ORDBMS 中,类型(TYPE)具有类(CLASS)的特征,可以看成类。

1. 行对象与行类型

一个行类型(ROW TYPE)可以使用如下语句定义:

```
CREATE ROW TYPE < row_type_name >
    ( < component declarations > );
```

行类型中属性类型可以是基本类型、扩展的关系类型。行类型也称为元组类型,因为它们的实例是表中的元组。

[例 3]

```
CREATE ROW TYPE Person_type                    /* 创建了一个行类型 Person_type */
      ( pno NUMBER,                            /* 以下是行类型属性说明 */
        name VARCHAR2(100),
        address VARCHAR2(100) );
```

然后,可以创建基于行类型的表,把类型实例化。定义语句如下:

```
CREATE TABLE < table_name > OF < row_type_name > ;
```

[例 4]

```
CREATE TABLE person_extent OF Person_type
  (pno PRIMARY KEY );                          /* 说明其中 pno 属性是主码 */
```

创建了一个表 person_extent,它由行类型 Person_type 来定义。表的属性定义直接对应行类型的属性定义。表中每行是一个对象,有 OID 标识。

2. 列对象与对象类型

在实际 ORDBMS 中(如 ORACLE)提供了列对象的概念,可以创建一个对象类型,表的属性可以是该对象类型。语句如下:

```
CREATE TYPE <type_name> AS OBJECT
    ( <component declarations> );
```

[例 5]

```
CREATE TYPE address_objtyp AS OBJECT
    (street VARCHAR2(50),
     city VARCHAR2(50) );
CREATE TYPE name_objtyp AS OBJECT
    (first_name VARCHAR2(30),
     last_name VARCHAR2(30) ) ;
```

然后创建一个表,定义其中的属性是对象类型。例如:

[例 6]

```
CREATE TABLE people_reltab (                    /* 这是传统的建表语句 */
    Id NUMBER(10),
    name_obj name_objtyp,                       /* 该列为对象类型 name_objtyp */
    address_obj address_objtyp);                /* 该列为对象类型 address_objtyp */
```

语法上看这和传统的建表语句类似。SQL3 扩展的是:允许表中的属性列是对象类型。

以上讨论的行类型和列对象提供了对象功能,还可以通过他们的各种组合来实现复杂对象类型的构造,但是它们并不提供 15.1.1 中所讨论的封装。而封装又是对象模型的一个本质特征,它由 SQL3 中的抽象数据类型来提供。

3. 抽象数据类型(Abastract Data Type,ADT)

SQL3 允许用户创建指定的带有自身行为说明和内部结构的用户定义类型称为**抽象数据类型**。定义 ADT 的一般形式为

```
CREATE TYPE <type_name> (
    所有属性名及其类型说明,
    [定义该类型的等于 = 和小于 < 函数,]
    定义该类型其他函数(方法));
```

ADT 有许多重要特点:

(1) ADT 的属性定义和行类型的属性定义类同。

(2) 在创建 ADT 的语句中,通过用户定义的函数比较对象的值。如果不定义该类型的比较函数,则采用默认的等于和小于函数来比较对象的大小。

(3) ADT 的行为通过方法(methods)、函数(functions)实现。在 SQL3 中函数和方法有很多不同。方法是和某个用户定义的类型紧密联系在一起的,而函数不是。函数和方法都可以用 SQL 语言或传统的程序设计语言(包括 JAVA)来书写。

(4) SQL3 要求抽象数据类型是封装的,而行类型则不要求封装。这是由数据库的特点决定的。数据库中数据是共享的,对共享数据的操作主要是查询、增、删、改,这是共同的隐

含的操作。不需要为共享数据另外定义一组操作，因此，没有必要自定义函数和封装。

对于大文本文档，图像数据，图形数据，时间序列数据，空间和地理数据等新的非结构化数据则需要定义各自特定的数据类型和操作，抽象数据类型较好地满足了这些新应用的需求。

由于特殊的数据类型一般和领域应用紧密相关，其中的函数常常是原来应用程序的一部分，因此一般由 ORDBMS 厂商和独立的软件开发商或有经验的用户部门合作，为特定应用开发类型库（或简称类库）。对于这些类库，Informix 称为数据刀片 Datablades，Oracle 称为 Cartridge，IBM DB2 称为 Extender，Sybase 称为 Plug-in 等。这些类库必须经过产品质量确认后才能作为产品的一部分发行。

（5）ADT 有 3 个通用的系统内置函数，构造函数（Constructor Function）、观察函数（Observer Function）和删改函数（Mutation Function）。

构造函数生成对象值，格式为 <ADT 名>()。例如，一个名为 Image_type 的 ADT，构造函数就是 Image_type()，它返回该类型的一个新对象（值）。

观察函数用来读取属性值。这是个隐含的函数，如果 park 是 Image_type 的一个对象，park(area)就返回 park 的属性 area 的值。

删改函数 Mutator，用来修改和删除属性值。与 UPDATE、DELETE 类似。

（6）ADT 可以参与类型继承。在类型继承过程中，子类型包括父类型的属性和方法，并可以增加自己的新的属性和方法。在 SQL3 中由根类型开始，与其相关的子类型、子类型的子类型构成一个类型层次（Type Hierarchies）。

ADT 的其他特点限于篇幅就不详细说明了。

15.2.3 参照类型（Reference Type）

SQL3 提供了一种特殊的类型：参照类型，也称为引用类型，简称 REF 类型。因为类型之间可能具有相互参照的联系，因此引入了一个 REF 类型的概念：

REF <类型名>

REF 类型总是和某个特定的类型相联系。它的值是 OID。OID 是系统生成的，不能修改。

下面通过例子来说明 REF 类型的有关概念。

[例 7]

```
CREATE ROW TYPE employee_type(
    name VARCHAR(35),
    age INTEGER );
CREATE ROW TYPE Comp_ type(
    compname VARCHAR(20),
    location VARCHAR(20) );
```

然后，可以创建基于行类型的表，如下所示：

```
CREATE TABLE Employee OF employee_type;
CREATE TABLE Company OF Comp_ type;
```

实际应用中 Employee 的元组与 Company 中的元组存在相互参照关系。即某个职工在某个公司工作。可以使用 REF 类型描述这种参照关系。

```
CREATE ROW TYPE Employment _type (
    employee REF (employee_type),
    company REF (Comp_ type) );
CREATE TABLE Employment OF Employment _type;
```

这样，表 Employment 中某一个元组的 employee 属性值是某个职工的 OID，company 属性值是该职工所在公司的 OID。从而描述了职工和公司相互的参照关系。

OID 可以被显式声明和访问。例如，[例 7] employee_type 的定义可以改为：

[例 8]

```
CREATE ROW TYPE employee_type(
    name VARCHAR(35),
    age INTEGER,
    emp_id REF(employee_type) );                    /* 声明 emp_id 是 REF 类型 */
```

然后可以使用 VALUES FOR < ref_attrribute > ARE SYSTEM GENERATED 由系统生成 OID 值，如[例 9]。

[例 9]

```
CREATE TABLE Employee OF employee_type
    VALUES FOR emp_id ARE SYSTEM GENERATED;
```

SQL3 提供了一种机制，它可以建立一个参照属性，指向基于该类型的一个指定的表，语句如下：

```
< 参照属性名 > [ REF( < 类型名 > ) | SCOPE IS < 关系名 >
```

例如，对于[例 5]中的地址对象可以先使用地址类型创建地址表，如[例 10]。然后修改[例 6]，在表定义中使用 REF 类型[例 11]。

[例 10]

```
CREATE TABLE address_objtab OF address_objtyp ;
```

[例 11]

```
CREATE TABLE people reltab2 (
    id NUMBER(4) PRIMARY KEY,
    name_obj name_objtyp,
    address_ref REF(address_objtyp) SCOPE IS address_objtab );
```

people_reltab2 表中的参照列 address_ref 存储的是所参照对象 address_objtyp 的 OID，SCOPE IS 用来限定所参照的对象在一个确定的表 address_objtab 的范围内。这样做的好处是，可以在参照列上建立索引，从而提高查询效率。

[例 12]

```
CREATE INDEX address_ref_idx ON people_reltab2(address_ref) ;
```

[例 13]

```
SELECT id
FROM people_reltab2 p
WHERE p. address_ref. city = '北京' and p. address_ref. street = '牛街';
```

15.2.4 继承性

ORDBMS 应该支持继承性，一般是单继承性。例如定义行类型 Person_type 的子类 emp_type，emp_type 就继承了它父类的属性，同时又可以定义 emp_type 子类自己的属性。这是人员类型没有而雇员才有的属性—雇员的 ID(EMP_ID)和工资(SALARY)。

[例 14]

```
CREATE TYPE emp_type                    /* 用户定义了一个 emp_type 类型 */
UNDER person_type AS(                   /* emp_type 是人员类型 person_type 的子类 */
    emp_id INTEGER,                     /* 定义 emp_type 类型的结构 */
    salary REAL )                       /* 定义 emp_type 类型的结构 */
NOT FINAL;
```

[例 11]还声明了该类型允许再有子类型，NOT FINAL 表示不是类层次结构中最后的“叶结点”，FINAL 则表示该类型是类层次结构的叶结点。

15.2.5 子表和超表

SQL3 支持子表和超表的概念。超表、子表、子表的子表也构成一个表层次结构。表层次和类型层次的概念十分相似。

如果一个基表是用类型来定义的，那么它可以有子表或/和超表。

[例 15] 对于下面的类型层次(图 15.4)，先定义这些类型 TYPE，然后创建基于这些类型的表。

```
CREATE TYPE person                      /* 创建 person 类型，这是根类型 */
    (id INTEGER,name VARCHAR(20),birthyear INTEGER,address VARCHAR(40))
    NOT FINAL;                          /* 这是根类型，NOT FINAL 表示可以有子类型 */
CREATE TYPE employee                    /* 创建 person 的子类型 employee */
    UNDER person                        /* 类型 employee 继承 person 的属性 */
    (salary INTECER)                    /* employee 定义自己的属性 */
```

```
    NOT FINAL;
CREATE TYPE executive                    /* 创建 employee 的子类型 executive */
    UNDER employee                       /* employee 的子类型 */
    (bonus INTEGER)
    FINAL;                               /* 类型 executive 下面没有子类型了 */
CREATE TYPE student                      /* 创建 person 的子类型 student */
    UNDER person
    (major VARCHAR(10),wage DECIMAL)
    FINAL;                               /* 类型 student 下面没有子类型了 */
```

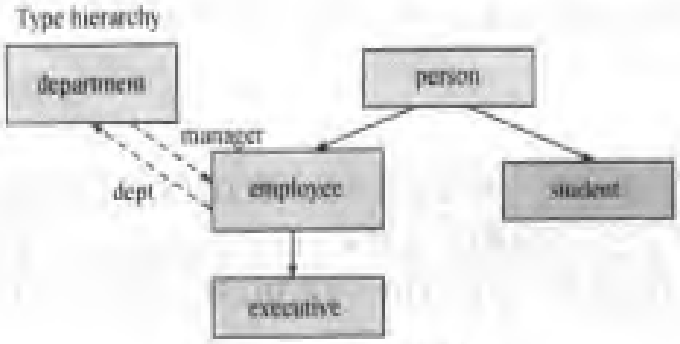


图 15.4 类型层次示例

以上 4 个类型构成了一个类型层次。

[例 16] Department 类型和 employee 具有相互参照的联系，可以使用 REF 来表示这种联系。

```
CREATE TYPE department                   /* 创建 department 类型 */
    (ID INTEGER,
    manager REF(employee),               /* 属性 manager 参照 employee 的 OID */
    Budget INTEGER);
ALTER TYPE employee                      /* 修改类型 employee */
ADD ATTRIBUTE dept REF(department);
                                         /* 属性 dept 参照 department 的 OID */
```

基于这些类型可以定义以下的基本表和表层次：

```
CREATE TABLE person_table OF person      /* 创建基于 person 类型的表 */
    (name WITH OPTIONS NOT NULL);
CREATE TABLE employee_table OF employee  /* 创建基于 employee 的表 */
    UNDER person_table;                  /* employee_table 表是 person_table 的子表 */
CREATE TABLE exec_table of executive
    UNDER employee_table                 /* exec_table 是 employee_table 的子表 */
CREATE TABLE student_table OF student
    UNDER person_table;                  /* student_table 是 person_table 的子表 */
CREATE TABLE dept_table OF department
```

```
    (manager SCOPE IS employee_table);
ALTER TABLE employee_table
    ALTER COLUMN dept ADD SCOPE IS dept_table;
```

这些表就构成了一个表层次。子表可以继承父表的属性、约束条件、触发器等,子表可以定义自己的新属性。

可以使用 SQL 的 SELECT,INSERT,DELETE,UPDATE 语句对这些表进行操作。对某个表的查询其实是对该表和它所有子表中对象集合的查询。

[例 17]

```
SELECT name,address
FROM person_table
WHERE birthyear <= 1970;
```

这个查询是要找出 person_table 表上 1970 年后出生的人。查询结果包括了 employee_table、exec_table、student_table 表上的所有人。

可以使用 ONLY"关闭"对子表的检索。办法是在 FROM 子句中使用 ONLY 将检索的对象限制为指定表中的对象,而不是该表和它的子表中的对象。

[例 18]

```
SELECT name,address
FROM ONLY person_table
WHERE birthyear <= 1970;
```

这个查询是只找出了 person_table 表上 1970 年后出生的人。不包括 employee_table、exec_table、student_table 表上的人。

对于 INSERT、DELETE、UPDATE 等其他操作,也有同样的规则。

INSERT:向子表插入一行时一般情况下会在该子表的超表上也插入一行。例如正常的向 student_table 表中插入一个新学生,则在 person_table 表上也插入一个新行。

DELETE:从表删除一行时一般情况下会在该表的超表和子表上也删除相应的一行。

15.3 小 结

本章首先介绍了 OO 模型的基本概念,然后讲解了 ORDB 所扩展的 OO 特征和功能。ORDB 在关系模型的基础上扩充了数据类型,支持对象概念和对象类型,支持用户定义的抽象数据类型 ADT,支持由基本数据类型、对象类型和用户自定义 ADT 类型构成的复杂对象,支持继承的概念,支持子表和超表的概念等。本章主要讲解 SQL3 所支持的对象特征和对象扩展。由于 ORDMS 的实现早于 SQL3 标准的制定,使得各个 ORDBMS 在支持对象模型所扩展的功能都要超过 SQL3,而且所采用的术语、语言语法又不尽相同。有关 ORDB 的数据操

纵语句，是对 SQL 语句的扩展，容易掌握。因篇幅有限这里就不专门讲解了。读者使用时可以参考具体的 ORDBMS 的语法说明和有关手册。

习 题

1. 定义并解释 OO 模型中以下核心概念：对象与对象标识、封装、类、类层次。

2. OO 模型中对象标识与关系模型中的"码"有什么区别？

3. 什么是单继承？什么是多重继承？继承性有什么优点？

4. 课程大作业：《对象－关系数据库应用开发》

（1）实验目的：通过实践，掌握 ORDB 的概念，掌握一个实际的 ORDBMS 软件的对象类型及其定义，掌握对象操作。

（2）基本要求：学习和熟悉某个 ORDBMS 软件，在此系统平台上，利用合适的开发工具开发一个 ORDB 销售订单系统。

销售订单系统的数据描述：顾客 Customer、库存 Stock、订单 Purchase Order、产品项 Item。一个顾客可以发出多个订单，一个订单包含多个产品项，每项产品对应有库存记录。设计这些对象合理的属性。其中顾客的地址是 Address(Street,City,Zip)。顾客的电话是 PhoneList(Phone1,Phone2,Phone3)。

a）用对象关系模型定义这些类型，顾客的地址用列对象来定义，顾客的电话用数组类型定义。

b）创建相应的对象表，定义具有的参照联系。

c）用 PL/SQL 或 C++ 或 JAVA 编程实现若干应用，可以包括以下几个方面。

插入：对建立的 ORDB 中的各个表，首先输入若干数据；

设计与开发：为销售订单系统设计若干功能，例如顾客定货，顾客管理，订单管理，库存管理等。实现若干子系统的功能。

查询：根据输入的订单号查出顾客和产品信息；根据输入的产品项查询它的库存量；查出北京的顾客信息；等等。

本章参考文献

1 Nelson M, Mattos. SQL3－新的 SQL 标准，新一代对象关系数据库. IBM 数据库通用技术，1999

2 Gulutzan P, Pelzer T. SQL－99 Complete. Really. Miller Freeman, Inc. USA, 1999

（该书全面描述 SQL3 标准，通过许多例子来说明 SQL3 的功能。对于 SQL3 来说，内容比较完整。该书的翻译本为：《SQL－3 参考大全》，齐舒创作室翻译，机械工业出版社，2000）

3 http://otn.oracle.com/products/oracle9i/content.html

4 Oracle9i Application Developer's Guide Object-Relational Features Release 2 (9.2) Part Number A96594－01

5 http://www－3.ibm.com/software/data/db2/udb/features.html

6 http://msdn.microsoft.com/library3

7 Kim W, Lochovsky F(Editors). Object-Oriented Concepts, Databases and Applications. Addison-Wesley, 1989

（本书讲解了面向对象数据库系统的基本概念、技术方法和应用。）

8 Stonbraker M, Rowe L. The Design of POSTGERS. Proceedings of the ACM SIGMOD, 1986

（本文介绍了 POSTGRES 系统，是众多介绍 POSTGRES 系统的论文之一。）

9 Kim W. Architectural Issues in Object-Oriented Databases. Journal of Object-Oriented Programming, March 1990

（本文介绍了 Orion 系统的各个方面，是众多介绍 Orion 系统的论文之一。）

10 Lecluse C, Richard P, Velez F. O_2: An Object-Oriented Data Model. Proceedings of the ACM SIGMOD, 1988

（本文介绍了 O_2 系统，介绍 O_2 系统的论文很多，这只是其中之一。）

11 Maier D, Stein J, et al. Development of an Object-Oriented DBMS. Proceedings of the Inter. Conference on Object-Oriented Programming Systems, Languages and Applications, 1986

（本文介绍了 Gemstone 系统，介绍 Gemstone 系统的论文很多，这只是其中之一。）

12 Fishman D, Beech D et al. IRIS: An Object-Oriented Database Management System. in Readings in Oriented-Oriented Database Systems, 1990

（本文介绍了 Iris 系统，介绍 Iris 系统的论文很多，这只是其中之一。）

13 Carey M L, DeWitt D et al. The EXODUS Extensible DBMS Project: An Overview, in Readings in Oriented-Oriented Database Systems. Morgan Kaufmann Publishers, 1990

（本文介绍了 EXODUS 系统，介绍 EXODUS 系统的论文很多，这只是其中之一。）

14 Hudson S E, King R. Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System. ACM TODS, Volume 14, Number 3, 1989

（本文介绍了 Cactis 系统）

15 王珊等. An Object-Oriented Model for GIS, 第三届泛太平洋计算机国际会议 PPCC－3 论文集, 1989

16 王珊. 面向对象的数据库系统. 计算机世界专题综述, 1990

17 王珊, 唐元昌, 任永杰等. OBMS/IDKE 的数据模型和无缝的编程语言. 计算机科学, No. 2, 1992

（本期的一组论文，全面介绍了中国人民大学数据与知识工程研究所研制的对象管理系统 OBMS/IDKE 的对象数据模型、无缝的编程语言、对象存储管理和动态模式管理。）

18 Stonbraker M M, Moore D. Object-Relational DBMSs: The Next Great Ware, Morgan Kaufmann Publisher, Inc. 1996

（本书作者是美国著名的数据库专家。本书已由杨冬青、唐世渭、裴芳等译校，《对象－关系数据库管理系统——下一个浪潮》，北京大学出版社出版，1997）

19 Malcolm Atkinson et al. The Object-Oriented Database System Manifesto. Proc. First International Conference on Deductive and Object-Oriented Databases. New York, N. Y.: Elsevier Science, 1990

（这篇文章试图为面向对象数据库系统（以及面向对象数据模型）作出定义。它描述了作为一个合格的面向对象数据库系统所应具备的主要特征和性质。）

20 中国人民大学数据与知识工程研究所编译: X3/SPARC/DBSSG/OODBTG 最终报告. 计算机工程与应用, No. 6, 1992

（本期是"OODBTG 最终报告"的专辑。1989 年 1 月，在美国 ANSI 所属 ASC/X3/SPARC 的数据库系统研究组（DBSSG）下，成立了面向对象数据库任务组（OODBTG），专门就对象数据管理（简称 ODM）为面向对象数据库管理系统（OODBMS）的标准化问题进行研究。该组织不是制定标准的技术委员会，而是研究如何实现新标准的工作组。OODBTG 的目标是对下面的问题进行调查、研究和取得共识：

(1) 提出能被大家接受的对象数据库的可行定义。

(2) 确定对象数据库技术与相关领域中面向对象技术之间的联系。这些相关领域包括程序设计语言、用户界面、方法学、信息建模方法等。

(3) 为将来在对象信息管理领域内的标准化工作建立一个框架。

该任务组于 1991 年 8 月发表了"OODBTG 最终报告"。该报告在大量调查研究的基础上,阐述了许多新的技术概念,提出了对象数据管理(ODM)参考模型和实现对象信息管理标准化的建议。上述工作对以后面向对象数据库系统的研究和系统开发产生了深远影响。)

第十六章 XML 数据库

随着 Web 应用的发展，越来越多的应用都将数据表示成 XML 的形式，XML 已成为网上数据交换的标准。所以当前数据库管理系统都扩展了对 XML 的处理，存储 XML 数据，支持 XML 和关系数据之间的相互转换。

由于 XML 数据模型不同于传统的关系模型和对象模型，其灵活性和复杂性导致了许多新问题的出现。在学术界，XML 数据处理技术成为数据库、信息检索及许多其他相关领域研究的热点，涌现了许多研究方向，包括 XML 数据模型、XML 数据的存储和索引、XML 查询处理和优化、XML 数据压缩等。

在工业界，各大数据库厂商都在产品中扩展了对 XML 的支持，并参与了 SQL/XML 标准的制定。

本章从数据管理的角度介绍 XML，介绍如何利用数据库技术存储和管理 XML 数据。重点介绍 SQL2003 标准对 XML 的扩展。有关 XML 文档格式定义等细节将从略从简，有兴趣的读者可以参考相关书籍和资料。

16.1 概 述

随着 Internet 的迅速发展，Web 上各种半结构化、非结构化数据源已经成为重要的信息来源。XML(eXtended Markup Language)意为可扩展的标记语言，用户可以定义自己的标记，用来描述文档的结构。XML 是 W3C(World Wide Web Consortium)在 1998 年制定的一项标准，用于网上数据交换。它是标准通用标记语言 SGML[ISO 8879](the Standard Generalized Markup Language)的一个子集。

在此之前，1989 年万维网(World Wide Web，简称 WWW 或 Web)的先驱，欧洲粒子物理实验室的 Tim Berners-Leek 以 SGML 风格的语法开发了超文本标记语言 HTML(HyperText Markup Language)。HTML 是 SGML 一个很小的子集，它简单易用，但是难以描述万维网上多样化的数据。

XML 正是在 SGML 和 HTML 的基础上发展起来的。XML 吸取了两者的优点克服了

SGML 过于复杂和 HTML 局限性等缺点。

相对于 HTML,XML 具有如下的一些特点:

1. 更多的结构和语义。XML 侧重于对文档内容的描述,而不是文档的显示。用户定义的标记描述了数据的语义,便于数据的理解和机器处理。HTML 只能表示文档的格式,而用 XML 可以描述文档的结构和内涵。

2. 可扩展性。允许用户自己定义标记和属性,可以有各种定制的数据格式。

3. 自描述性。对数据的描述和数据本身都包含在文档中,使数据具有很大的灵活性。

4. 数据与显示分离。XML 所关心的是数据本身的语义,而不是数据的显示,所以可以在 XML 数据上定义多种显示形式。文档内容与展示格式分离。展示格式用一个独立的格式单(stylesheet)定义。

5. 简洁性。与标准通用标记语言 SGML 相比,XML 简单易用。

正是由于这些特点,尤其是可以自由扩展的标记和独立于文档的格式单,使 XML 得以推广应用。

16.2 XML 简 介

下面简单地介绍 XML 文档常用的语法成分。XML 的语法简单易学,其中的规则不难理解。图 16.1 给出了一个 XML 文档的例子。

一、XML 说明

XML 说明(XML declaration)是对 XML 文档处理的环境和要求的说明。XML 说明必须在文档的第一行,例如:

```
<? xml version = "1.0" encoding = "UTF-8" standalone = "no"? >
```

xml version = "1.0",说明使用的 XML 版本号,其中的字母是区分大小写的。

encoding = "UTF-8"(Unicode Transformation Formal-8),文字编码说明。位于版本属性之后,指出文档是使用何种字符集建立的。默认值是 Unicode 编码(UTF-8 或 UTF-16)。

standalone = "no",独立文档说明。使用的属性值可以为"yes"或"no"。"yes" 表示 XML 文档中的所有的实体声明都必须包含在该文档内部,如果是"no"表示需要引用外部的标记声明。

二、元素

元素(element)是 XML 文档的主要组成部分。图 16.1 中 pub、library、book 等都是元素。元素有名字,即标记名。起始标记的形式是 <标记名>,终止标记的形式是 </标记名>,如 <pub> </pub>、<library> </library>。

元素可以嵌套,图 16.1 中 book 嵌套在 library 中、library 嵌套在 pub 中。元素中还可以

包括处理指令、注释、字符数据(CDATA)段或者字符。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<pub>
    <library>Beijing Library</library>
    <book year="2000">
        <title>Database System Concepts</title>
        <price>26.50</price>
        <author id="101">
            <name>Kaily Jone</name>
        </author>
        <author id="102">
            <name>Silen Smith</name>
        </author>
    </book>
    <book year="2001">
        <title>Introduction to XML</title>
        <price>18.80</price>
        <author id="103">
            <name>Kaily Jone</name>
        </author>
    </book>
    <article editorID="105">
        <title>A Query Language for XML</title>
        <author id="104">
            <name>Kaily Jone</name>
        </author>
    </article>
</pub>
```

图 16.1 一个 XML 文档示例

XML 文档必须有且只有一个根元素,XML 文档的第一个元素就是根元素。

XML 中的元素名称是区分大、小写的,它必须开始于字母或下划线(_),后面可以跟任意长度的字母、数字、句点(.)、连接符(-)、下划线或冒号。

三、属性

属性(attribute)通常用来描述元素的有关信息。属性名和属性值在元素的起始标记中给出,格式为:<元素名 属性名 = "属性值" >,如 <book year = "2000" >。属性值必须出现在单引号或双引号中。一个元素可以有任意多个属性,每个属性取不同的属性名。

四、处理指令

处理指令(processing instructions)是为使用一段特殊代码而设计的标记。处理指令通常用来为处理 XML 文档的应用程序提供信息。这些信息包括如何处理文档、如何显示文档

等。

处理指令由两部分组成:处理指令名称和数据,其格式为 <? target data? >。例如 <? cocoon-process type = "sql"? > 、<? display table-view? >。处理指令可以在元素说明中给出也可作为 XML 文档的顶层结构放在根元素的前面或后面。

五、注释

XML 中可以有注释(comments)。注释以 <!- 开始,以 -> 结束,这两个字符序列之间是注释的文本内容。注释可以在 XML 文档的任何地方插入。

六、良构的 XML 文档

XML 文档应该是良构的(well-formed,也称为格式正确的)。所谓良构的 XML 文档是指:文档的构造从语法上都是正确的;只有一个顶层元素,即根元素;至少包含一个元素,即文档中必须有根元素;所有的起始标记都有与之对应的终止标记,或者使用空元素速记语法;所有的标记都正确的嵌套;每一个元素的所有属性具有不同属性名。

图 16.1 是一个良构的出版物 XML 文档。它有一个 pub 根元素,该元素中包含的子元素有 library、book 和 article;每个 book 元素中又有子元素 title、price、author(一个或多个),book 元素中有属性 year;类似的每个 article 元素中有子元素 title、author 和属性 editor ID 等等。

七、实体

XML 文档中对于重复使用的文档内容可以用实体(entities)来定义。格式为:<! ENTITY 实体名"实体内容" > 。例如 <! ENTITY DW "Data Warehouse" >。当 XML 处理器遇到字符串 &DW 时就用字符串 Data Warehouse 代替该实体 DW,即引用实体的格式为 & 实体名。

16.3 XML 数据模型

XML 数据是自描述的、不规则的,可以用图模型来表示,例如,可以使用图 16.2 表示图 16.1 中的 XML 数据。

其中,图的边标记为元素的标识名,结点标记为属性 - 值对应的集合,叶子结点代表元素的文本内容。XML 图要有一个根结点,每个结点用唯一的对象标识符表示,为了降低图的复杂度,上图省略了结点的对象标识符。

XML 数据与半结构化数据非常类似,可以看成是半结构化数据的特例。但它们之间还是存在着一些差别,使得半结构化数据模型不能很好地描述 XML 的特征。

(1) XML 中存在参照。XML 的元素可以用类型为 ID 的特殊属性来标识,而对元素的参照则使用类型为 IDREF 的属性来定义,引用要参照的元素的标识。

(2) XML 中的元素是有序的。

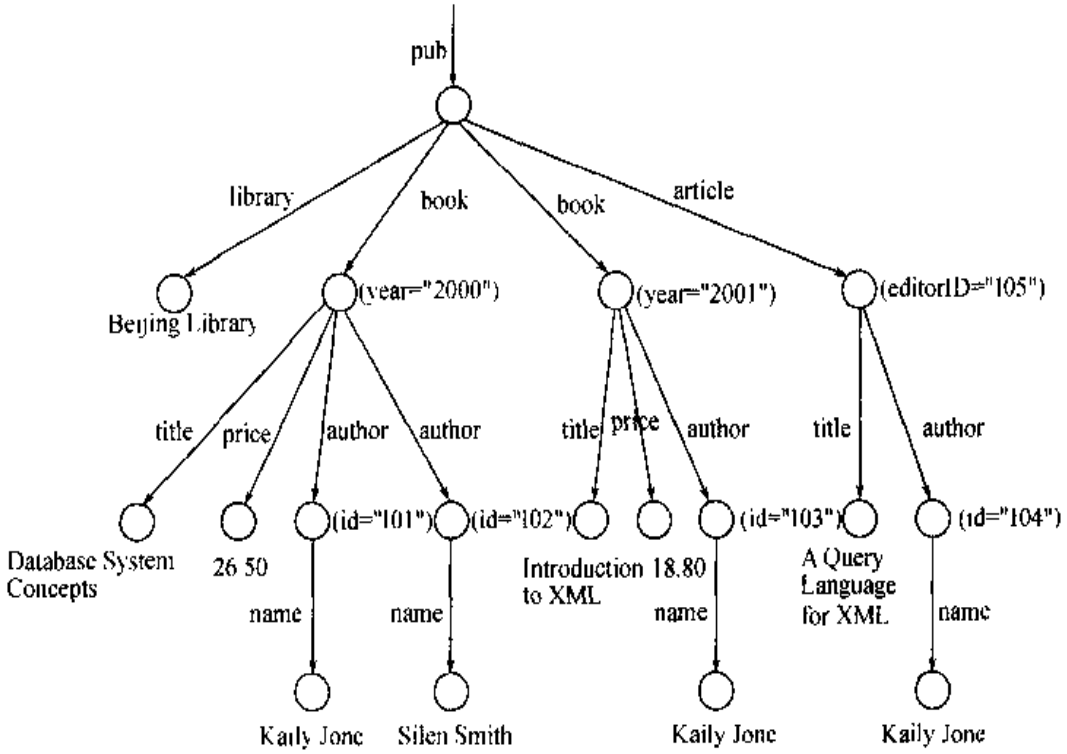


图 16.2 图 16.1 中的 XML 数据的图模型表示

(3) XML 中可以将文本与元素混合。XML 中的元素可以包含文本和子元素的混合。

(4) XML 包含许多其他的内容:处理指令、注释、实体、CDATA、文档类型定义(DTD)等。

这些差别是由于 XML 是一种文档标记语言而产生的。正是由于 XML 具有不同于半结构化数据的自身特征,所以需要研究新的 XML 数据模型来描述 XML 数据。

目前还没有公认的 XML 数据模型,W3C 已经提出的有:XML Information Set,Xpath1.0 Data Model,DOM model 和 XML Query Data Model。这四种模型都采用树结构。表 16.1 给出了它们之间的比较。在这四种模型中,XML Query Data Model 是较为完全的一种。

表 16.1 XML 数据模型的比较

	XML Information Set	Xpath 1.0 Data Model	DOM 1.0 Level 2	XML Query Model
对象	XML 文档	XML 文档	XML(或 HTML)文档	XML 文档或部分的集合
定义方式	对 XML 语法项的附加描述	对一组结点类型的数据结构和字符串值的描述	一组对象接口的描述	Constructor 和 accessor 的功能描述
是否要求 DTD 或 XML Schema 验证	否	否	否	是

16.3.1 XML DTD

在实际的应用中,为了充分发挥XML的潜力,实现数据的自动处理,往往需要事先约定模式。在XML标准中提供了文档类型定义DTD(Document Type Descriptors),用来描述XML文档的结构,类似于模式的概念。

DTD在XML标准中是可选的,不具有强制性。许多应用制定了领域性的DTD标准,以实现在各自领域中的数据交换和自动处理。

DTD定义在XML文档中出现的元素、属性,这些元素出现的次序、出现的次数,如何相互嵌套以及XML文档结构的其他详细信息。

一、元素的定义

DTD定义元素的格式为:<! ELEMENT 元素名(元素内容描述) >

[例1] <! ELEMENT name (title?,first-name,last-name) >

定义了元素name中包括了title、first-name、last-name等元素。问号表示这一元素是可选的,可出现一次或不出现。

[例2] <! ELEMENT addressbook (address +) >

加号表示该元素(address)必须出现一次,可以出现多次。

[例3] <! ELEMENT private-addresses (address *) >

星号表示address元素可以出现0或多次。

[例4] <! ELEMENT name(title?,first-name ,(middle-initial | middle-name)?,last-name) > 竖线符号表示从列表中选择一个。这里就是从middle-initial和middle-name列表中选择一个。

[例5] <! ELEMENT city (#PCDATA) >

PCDATA表示元素city中不嵌套其他元素,只包括文本数据。

二、属性的定义

DTD中定义属性的格式为:

<! ATTLIST 元素名 (属性名 属性类型 缺省声明) * >

属性类型有字符串类型、枚举类型等。缺省声明有3种:#REQUIRED,表示该属性在XML文件中是必须出现的;#IMPLIED,表示该属性在XML文件中是可以缺省的;三是声明缺省属性值。

[例6] <! ATTLIST book year CDATA #IMPLIED >

定义book元素的属性year,类型是CDATA,即字符串类型,#IMPLIED属性year可以缺省。

通常把DTD存储在一个后缀为.dtd的外部文件里。该外部DTD文件可以被多个XML文件所共享。这样只要写一个DTD文件,就可以让多个XML文件引用。

[例 7] 一个 XML DTD 示例

```
<!- address. dtd ->
<! ELEMENT address (name, street, city, state, postal-code) >
<! ELEMENT name (title? first-name, last-name) >
<! ELEMENT title (#PCDATA) >
<! ELEMENT first-name (#PCDATA) >
<! ELEMENT last-name (#PCDATA) >
<! ELEMENT street (#PCDATA) >
<! ELEMENT city (#PCDATA) >
<! ELEMENT state (#PCDATA) >
<! ELEMENT postal-code (#PCDATA) >
```

16.3.2 XML Schema

尽管 DTD 可以定义 XML 文档结构,但对于许多 XML 应用来说还是不够的。例如,DTD 不支持对参照关系的约束;对元素出现次数的定义不够精确,可能产生模糊的定义;DTD 几乎没有数据类型的定义,一切都是基于字符串;DTD 的定义并不符合 XML 语法等等。为了更好地利用文档的结构,采用与 XML 文档相同的形式来定义文档的结构,增加对数据类型的支持,由此 W3C 提出了定义 XML 模式的另外两个标准 XML Schema 和 Document Content Descriptors(DCDs),它们是对 DTD 的扩展。

XML Schema 用 XML 来定义其文档的模式,支持对结构和数据类型的定义,下面是一个 XML Schema 的例子:

[例 8]

```
< elementType name = " paper" >
        < sequence >
                < elementTypeRef name = " title" / >
                < elementTypeRef name = " author" minOccurs = "0" / >
                < elementTypeRef name = " year" / >
                < choice >
                        < elementTypeRef name = " journal" / >
                < elementTypeRef name = " conference" / >
                < /choice >
        < /sequence >
< /elementType >
```

表 16.2 列出了 DTD 和 XML Schema 之间的特征比较,可以看出 XML Schema 是更为完善的模式定义语言。尽管 DTD 非常简单,许多特征都不支持,但它简便易用,成为目前使用最广泛的 XML 模式标准。

表 16.2 DTD 和 XML Schema 的特征比较

	Syntax in XML	Namespace	Include & Import	Built-in type	User-defined type	Domain constraint	Explicit Null
DTD	No	No	No	10	No	No	No
XML Schema	Yes	Yes	Yes	37	Yes	Yes	Yes

把遵守 XML 语法,但没有 DTD 或 XML Schema 的 XML 文档称为良结构(well-formed)的文档。既遵守 XML 语法规则也符合某一给定模式的 XML 文档称为有效(validXML)的文档。定义 XML 的模式的语言有多种,例如上面介绍的 DTD 和 XML Schema。如果开发人员已经在 DTD 或模式中定义了文档结构,而某个文档没有遵守那些规则,那么这个文档是无效的文档。

16.4 SQL/XML

上面简单介绍了 XML 文档的基本格式及 XML 数据模型和模式等有关概念。下面介绍 SQL2003 标准中对 XML 的支持。

由于 XML 的日益普及,各大数据库厂商纷纷实现了对 XML 的支持,但方法、术语、语言都不统一。为此 SQL2003 标准扩展了 SQL 语法,增加了对 XML 的支持,定义了数据库语言 SQL 与 XML 结合的方式,扩展的部分(SQL2003 的第 14 部分)称为 SQL/XML[①]。表 16.3 列出了 SQL/XML 中的主要关键词及其功能分类。

表 16.3 SQL/XML 中的主要关键词及功能分类

功能		SQL/XML 中的关键词
数据类型定义		XML
强制数据类型转化		XMLCAST
字符串→XML		XMLPARSE
XML→字符串		XMLSERIALIZE
XML 发布函数	关系数据→XML	XMLELEMENT,(XMLNAMESPACES,XMLATTRIBUTES),XMLFOREST,XMLCONCAT,XMLAGG,XMLCOMMENT,XMLPI
XML 提取函数	XML→XML	XMLQUERY
	XML→关系	XMLTABLE
	XML→布尔值	XMLEXISTS

SQL/XML 定义了新的数据类型—XML 数据类型,以及一组函数。这组函数实现了在

① 见本章参考文献[1]。

SQL 中对 XML 的操作，如 XML 的构造和抽取，XML 数据和关系数据之间的相互转换等。使得 XML 数据可以存储在关系数据库中，可以用标准的查询语言 XQuery 从中提取信息，也可以将关系数据转换成 XML 的形式呈现给用户，从而实现了 XML 与关系数据的双向对称操作。如图 16.3 所示。

下面主要介绍 SQL/XML3 个方面的内容：XML 数据类型、XML 发布函数和 XML 提取函数。限于篇幅，这里只通过例子讲解最基本的内容。

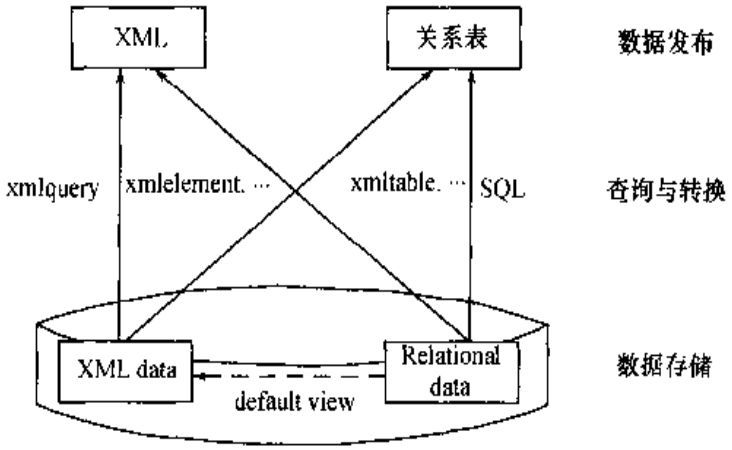


图 16.3 XML 和关系数据间的双向转换

16.4.1 XML 数据类型

SQL/XML 定义了一个原生(Native)SQL 数据类型：XML 数据类型，用“XML”来表示。XML 数据类型与 INTRGER、DATE、CLOB 等数据类型一样，都是 SQL 数据类型的一种，可以使用这种新的数据类型来定义变量、参数以及关系的列等。当使用 XML 数据类型定义关系的列时，关系数据库就可以直接将 XML 当成单独的一列保存起来。

[例 1] 创建一个书店表 bookstore，其中 bib 列的类型是 XML。

```
CREATE TABLE bookstore(id INT PRIMARY KEY,bib XML);
```

[例 2] 可以将一个 XML 文档加入书店 bookstore 中。

```
INSERT INTO bookstore
VALUES (1492,
    xmlparse ('<? xml encoding = "UTF-8"? >
            <book price = "25.00" >
                <lang > English </lang >
                …
            </book >') );
```

其中 XMLParse 函数将一个字符串作为 XML 解析，返回相应的 XML 结构。反之，XMLSerialize 函数将 XML 数据类型值转化为字符串数据类型值。

XML 数据类型的引入使得系统可以实现 XML 数据的内部表示，而不是仅仅使用文本形式的 XML 数据。

使用 XMLCast 可以将某种 SQL 预定义类型的数据转换为一种 XML 类型，或者将某种 XML 类型的数据转换为其他的 SQL 预定义类型或 XML 类型。它的格式与一般的 SQL CAST 相同。

```
XMLCAST ( value-expression AS type )
```

16.4.2 XML 发布函数

SQL/XML 允许在 SQL 表达式中使用 XML 发布函数从关系数据创建 XML 结构，SQL/XML 提供了一组发布函数。如表 16.3 所示，一共 8 个。使用这组函数可以用来构造 XML 数据。

一、XMLElement 函数

XMLElement()创建一个 XML 元素，可以指定元素名字。

[例 3]

```
SELECT e.id,XMLELEMENT(NAME "Emp",e.fname || ' ' || e.lname)
AS "result"                          /* 关系中查询结构发布为 XML，元素名为 Emp */
FROM employees e;                    /* 元素内容为 e.fname || ' ' || e.lname */。
```

XMLElement 函数的第 1 个参数是要构建的元素名，接下来可以是 namespace 的申明以及该元素的属性(如果不存在，则不需要给出)，然后是要构建的元素内容。

假设关系表 employees 如下：

id	fname	lname	hire	dept
1001	John	Smith	2000-05-24	Accounting
1206	Bob	Martin	1996-02-01	Shipping

查询返回的结果是：

ID	Result
1001	<Emp>John Smith</Emp>
1206	<Emp> Bob Martin</Emp>

XMLElement 也可以嵌套使用，用来构建 XML 层次结构。

[例 4]

```
SELECT e.id,XMLELEMENT( NAME "Emp",
                XMLELEMENT (NAME "name",e.fname ),
                XMLELEMENT (NAME "hiredate",e.hire )
) AS "result"
FROM employees e;
```

查询返回的结果是：

ID	result
1001	< Emp > < name > Smith < /name > < hiredate > 2000 - 05 - 24 < / hiredate > < /Emp >
1206	< Emp > < name > Martin < /name > < hiredate > 1996 - 02 - 01 < /hiredate > < /Emp >

XMLElement 函数的参数还可以是一个子查询：

[例 5]

```
SELECT e. id,XMLELEMENT ( NAME "Emp",
                XMLELEMENT (NAME "name",e. lname),
                XMLELEMENT (NAME "dependants",
                (SELECT COUNT ( * )
                FROM dependants d
                WHERE d. parent = e. id) )
        ) AS "result"                    /* 子查询的结果是元素 dependants 的内容 */
FROM employees e;
```

返回的结果可能是：

ID	result
1001	< Emp > < name > Smith < /name > < dependants > 3 < /dependants > < /Emp >

二、XMLAttributes 函数

XMLAttributes 函数为构建的元素生成一组属性。该函数仅在 XMLElement 函数中使用。如果没有给出属性名，则隐含使用列名。

[例 6]

```
SELECT e. id,XMLELEMENT (NAME "Emp",
                XMLATTRIBUTES (e. id,e. lname AS "name")
    ) AS "result"
  FROM employees e;
```

返回的结果可能是：

ID	result
1001	< Emp ID = "1001" name = "Smith" / >
1206	< Emp ID = "1206" name = "Martin" / >

这里在元素 Emp 中创建了两个属性:ID 和 name,第一个属性名是从 ID 列的名字隐含得到的;第二个属性名是显式的从"name"得到。

三、XMLNamespaces

函数 XMLNamespaces 用来给出名空间 Namespaces。

[例 7]

```
SELECT e. id,XMLELEMENT( NAME "admi:employee",
                    XMLNAMESPACES('http://www. admi. com' AS "admi"),
                    XMLATTRIBUTES( e. workdept AS "admi:department"),
                    e. lastname
        ) AS "result"
    FROM employees e
    WHERE e. job = 'ANALYST';
```

可能的结果是:

ID	result
1130	< admi:employee xmlns:admi = http://www. admi. com admi:department = "C01" > QUINTANA < /admi:employee >
1140	< admi:employee xmlns:admi = http://www. admi. com admi:department = "C01" > NICHOLLS < /admi:employee >

四、XMLForest 函数

XMLForest 函数用来生成一组元素。每个参数生成一个 XML 元素,元素名可以显式地给出,也可以隐含地使用列名。

[例 8]

```
SELECT e. id,XMLFOREST ( e. hire,e. dept AS "department") AS "result"
FROM employees e;
```

结果是:

ID	result
1001	< HIRE > 2000 - 05 - 24 < /HIRE > < department > Accounting < /department >
1206	< HIRE > 1996 - 02 - 01 < /HIRE > < department > Shipping < /department >

XMLFOREST 函数的参数 e. hire 和 e. dept 都生成一个 XML 元素,第一个元素未指定名字,就使用列名 HIRE 作为元素名。

五、XMLConcat 函数

XMLConcat 函数将两个以上的 XML 数据值串接起来。

[例 9]

```
SELECT e.id,XMLCONCAT (
                XMLELEMENT (NAME "first",e.fname),
                XMLELEMENT (NAME "last",e.lname)
              ) AS "result"
FROM employees e;
```

返回的结果可能是：

ID	Result
1001	<first>John</first> <last>Smith</last>
1206	<first>Mary</first> <last>Martin</last>

如果某个结果返回空值，则不出现在串接中。

六、XMLAGG 函数

XMLAGG 是一个聚集函数。它的参数是一个 XML 数据类型的表达式。对于每个分组，其中所有元组执行该表达式所产生的结果将被串接起来形成 XML 数据，作为这个分组的结果返回。

[例 10]

```
SELECT XMLELEMENT (NAME "Department",
          XMLATTRIBUTES (e.dept AS "name"),
          XMLAGG (XMLELEMENT (NAME "emp",e.lname)) )
    AS "dept_list",
       COUNT( * ) AS "dept_count"
FROM employees e
GROUP BY dept;
```

这个查询根据 dept 分组，每一组中的每一行都由 XMLELEMENT (NAME "emp",e.lname) 生成 XML 数据类型值，然后作为 XMLAGG 的参数串接起来形成一组 XML 数据类型值。

假设关系表 employees 加上了 2 条记录，共 4 条记录：

id	fname	lname	hire	dept
1000	Tom	Yates	2001-08-02	Accounting
1001	John	Smith	2000-05-24	Accounting
1205	Mary	Oppenheimer	2000-01-19	Shipping
1206	Bob	Martin	1996-02-01	Shipping

返回的结果是：

dept_list	dept_count
< Department name = " Accounting" > < emp > Yates < /emp > < emp > Smith < /emp > < /Department >	2
< Department name = " Shipping" > < emp > Oppenheimer < /emp > < emp > Martin < /emp > < /Department >	2

XMLAGG 函数中还可以使用 ORDER BY 子句指定返回值之间的串接顺序。

[例 11]

```
SELECT XMLELEMENT (NAME "Department",
        XMLATTRIBUTES (e.dept AS "name"),
        XMLAGG (XMLELEMENT (NAME "emp",e.lname)
                ORDER BY e.lname)
      ) AS "dept_list",                    /* 在每个 dept 组中根据 lname 排序 */
        COUNT( * ) AS "dept_count"
FROM employees e
GROUP BY dept;
```

返回的结果是：

dept_list	dept_count
< Department name = " Accounting" > < emp > Smith < /emp > < emp > Yates < /emp > < /Department >	2
< Department name = " Shipping" > < emp > Martin < /emp > < emp > Oppenheimer < /emp > < /Department >	2

七、XMLComment 函数

XMLComment 用于生成注释。

[例 12]

```
XMLCOMMENT('This is a comment')
```

结果生成 <!- This is a comment -> 。

八、XMLPI 函数

XMLPI 函数用于生成处理指令。

[例 13]

```
XMLPI(NAME "includeFile",'/POs/template.hls')
```

结果生成 <? includeFile /POs/template.hls? >。

16.4.3 XML 提取函数

一、XMLQuery 函数

XQuery 是 XML 数据的查询语言。SQL/XML 提供将 XQuery 嵌入 SQL 的机制,对存储在关系数据库中的 XML 数据可以用 XQuery 进行查询。XMLQuery 函数返回的数据类型为 XML。

[例 14] 选出价格低于 80 的所有英文书,返回价格和作者名。

```
SELECT XMLQUERY(
          'for $bib in $doc/bib[lang/text( ) ='English'],
              $book in $bib//book
WHERE $book/@price < 80
RETURN <bookInfo>
              {$book/@price},{$book/author/name}
           </bookInfo>'
PASSING bib AS "doc" RETURN CONTENT) AS "result"
FROM bookstore;
```

返回的结果可能是:

result
<bookinfo price = "25" > <name> Serge Abiteboul </name> </bookinfo>
<bookinfo price = "60" > <name> Roger Penrose </name> </bookinfo>

注意 Xquery 中的 return 子句有构造 XML 的功能。

二、XMLTable 函数

XMLTable 函数从 XML 数据中提取信息,返回结果是一个二维的关系表(列的定义用 XPath 路径表达式)。

[例 15] 选出价格低于 80 的英文书,显示书名、作者名、出版年份和价格(返回关系表的格式)。

```
SELECT T.title,T.author,T.year,T.price
FROM bookstore AS B,
  XMLTABLE(
    'for $bib in $doc/bib[lang/text( ) ='English'],
```

```
        $book in $bib//book
    WHERE $book/@price < 80
    RETURN $book'
    PASSING B.bib AS "doc"
    COLUMNS "title" VARCHAR(100)   PATH './title',
            "year" INTEGER         PATH './year',
            "price" DOUBLE         PATH './@price',
            "author" VARCHAR(30)   PATH './author/name'
) AS T;
```

返回的可能结果是：

title	author	year	price
Data on the Web	Serge Abiteboul	1999	15
The Emperor's New Mind	Roger Penrose	1989	60

三、XMLExists

XMLExists 是一个谓词，用来检查 XQuery 查询在 XML 数据上的执行结果是否为非空。如果查询结果是一个空序列，则谓词返回假，否则为真。

[例 16] 查询标了价格的所有书目。

```
SELECT id,bib
FROM bookstore
WHERE XMLEXISTS('$doc/bib/book/@price' PASSING bib AS "doc");
```

可以看到 SQL/XML 提供了机制完成 XML 和关系数据间的各种转换。各大数据库厂商 Oracle、DB2、SQL Server、Sybase 都参与了 SQL/XML 标准的制定，正致力于完善对 XML 的支持，实现 SQL/XML 的部分或全部函数。

16.5 小 结

本章从数据管理的角度介绍 XML。重点介绍 SQL2003 标准对 XML 的扩展 SQL/XML。SQL/XML 基于关系模型，允许在 SQL 查询中直接使用 XML 发布函数生成 XML 结构，对传统的 SQL 做了扩展，对 SQL 编程者来说易学易懂。同时 SQL 是一种成熟的语言，数据库厂商在 SQL 的查询优化上已经积累了多年的经验，且已存在很多 API，如 ODBC、JDBC 等，使得 SQL/XML 具有其他 XML 查询标准如 XQuery 没有提供的特性，如 SQL/XML 可以使用 JDBC 返回结果。可以说 SQL/XML 既提供了从关系数据创建 XML 的功能，又能很好的适应已有的 SQL 环境。

习　题

课程大作业：在一个支持 XML 的 DBMS 平台上设计实现一个 XML 数据库应用系统。以网上书店系统为例（你也可以实现其他类似的系统）。

（1）给出应用需求说明。

（2）实现的主要功能有：

用户——通过网上订书，购书；

DBA——管理数据库；

对 XML 数据库的查询、增删改；

XML 数据与关系数据的对称存取等。

（3）数据库中存放用户信息、用户的历史订单信息、图书书目信息。用户信息、用户的历史订单信息存在关系表中，图书书目信息以 XML 形式存储。

（4）用户可以在网上书店注册、查询书目、订书，结果以 XML 形式显示。开始时只显示作者、书名、价格、出版社；然后修改显示结果，增加显示简介、版权页，其中版权页包含：出版社、出版年月、责任编辑、字数。

（5）DBA 可以用 SQL 来查询用户的历史订单和书目信息，也可以用 XML 语句来查询用户的历史订单和书目信息，实现 XML 和关系的对称存取。

本章参考文献

1 SQL/XML 2003, the first edition of the SQL/XML standard published by the ISO as part 14 of the SQL standard: ISO/IEC 9075 - 14:2003.

2 Eisenberg A, Melton J. Advancements in SQL/XML. SIGMOD Record Vol. 33 No. 3, 2004

3 Krishnaprasad M, Liu Z H, etc.. Towards an industrial strength SQL/XML Infrastructure. Proceedings of the ICDE 2005, 2005

4 http://www.sqlx.org/

5 XML. http://www.w3.org/XML/.

6 XPath. http://www.w3.org/TR/xpath.

7 XQuery. http://www.w3.org/XML/Query

8 exist. http://www.exist-db.org, An Open Source Native XML Database

9 Liu Z H, Krishnaprasad M, Arora V. Native XQuery Processing in Oracle XMLDB. Proceedings of the SIGMOD2005, 2005

10 Nicola M, van B der Linden. Native XML Support in DB2 Universal Database. Proceedings of the VLDB 2005, 2005

第十七章 数据仓库与联机分析处理技术

数据仓库(Data Warehouse,DW)是信息领域中近年来迅速发展起来的数据库新技术。数据仓库的建立,能充分利用已有的数据资源,把数据转换为信息,从中挖掘出知识,提炼成智慧,最终创造出效益。所以,越来越多的企业开始认识到数据仓库应用所带来的好处。

计算机系统中存在着两类不同的数据处理工作:操作型处理和分析型处理,也称作OLTP(联机事务处理)和OLAP(联机分析处理)。

操作型处理也叫事务处理,是指对数据库联机的日常操作,通常是对一个或一组记录的查询和修改,例如火车售票系统、银行通存通兑系统、税务征收管理系统等。这些系统要求快速响应用户请求,对数据的安全性、完整性以及事务吞吐量要求很高。

分析型处理,是指对数据的查询和分析操作,通常是对海量的历史数据查询和分析,例如金融风险预测预警系统、证券股市违规分析系统。这些系统要访问的数据量非常大,查询和分析的操作十分复杂。

两者之间的差异,使得传统的数据库技术不能同时满足两类数据的处理要求,数据仓库技术就应运而生了。

17.1 数据仓库技术

数据仓库和数据库只有一字之差,似乎是一样的概念,但实际则不然。数据仓库是为了构建新的分析处理环境而出现的一种数据存储和组织技术。由于分析处理和事务处理具有极不相同的性质,因而两者对数据也有着不同的要求。数据仓库概念的创始人W. H. Inmon在其《Building the Data Warehouse》一书中,列出了操作型数据与分析型数据之间的区别,具体如表17.1所示。

表 17.1 操作型数据和分析型数据的区别

操作型数据	分析型数据
细节的	综合的,或提炼的
在存取瞬间是准确的	代表过去的数据
可更新	不更新
操作需求事先可知道	操作需求事先不知道
生命周期符合 SDLC	完全不同的生命周期
对性能要求高	对性能要求宽松
一个时刻操作一元组	一个时刻操作一集合
事务驱动	分析驱动
面向应用	面向分析
一次操作数据量小	一次操作数据量大
支持日常操作	支持管理决策需求

基于上述操作型数据和分析型数据之间的区别,可以给出数据仓库的定义:数据仓库是一个用以更好地支持企业(或组织)决策分析处理的、面向主题的、集成的、不可更新的、随时间不断变化的数据集合。数据仓库本质上和数据库一样,是长期储存在计算机内的、有组织、可共享的数据集合。

一、数据仓库的基本特征

数据仓库和数据库主要的区别是数据仓库中的数据具有以下四个基本特征。

- 数据仓库的数据是面向主题的;
- 数据仓库的数据是集成的;
- 数据仓库的数据是不可更新的;
- 数据仓库的数据是随时间不断变化的。

1. 主题与面向主题

数据仓库中的数据是面向主题进行组织的。主题是一个抽象的概念,是在较高层次上将企业信息系统中的数据综合、归类并进行分析利用的抽象;在逻辑意义上,它对应企业中某一宏观分析领域所涉及的分析对象。比如一家商场,概括分析领域的对象,应有的主题包括供应商、商品、顾客等。面向主题的数据组织方式是根据分析要求将数据组织成一个完备的分析领域,即主题域。

主题域应该具有以下两个特点。

① 独立性,如针对商品进行的各种分析所要求的是“商品”主题域,它必须具有独立内涵。

② 完备性,任何对商品的分析处理要求,应该能在“商品”这一主题内找到该分析处理所要求的内容;如果对商品的某一分析处理要求涉及现存“商品”主题之外的数据,那么就应当将这些数据增加到“商品”主题中来,从而逐步完善“商品”主题。或许有人担心,要求

主题的完备性会使得主题包含有过多的数据项而显得过于庞大。这种担心是不必要的，因为主题是一个逻辑上的概念，如果主题的数据项多了，实现时可以采取各种划分策略来化大为小。

主题是一个在较高层次上对数据的抽象，这使得面向主题的数据组织可以独立于数据的处理逻辑，因而可以在这种数据环境上方便地开发新的分析型应用；同时这种独立性也是建设企业全局数据库所要求的，所以面向主题不仅适用于分析型数据环境的数据组织方式，同时也适用于建设企业全局数据库的组织。

2. 数据仓库是集成的

前面已经讲到，操作型数据与分析型数据之间差别甚大，数据仓库的数据是从原有的分散的数据库数据中抽取来的，因此数据在进入数据仓库之前，必然要经过加工与集成，统一与综合。这一步实际是数据仓库建设中最关键、最复杂的一步。

首先，要统一原始数据中所有矛盾之处，如字段的同名异义、异名同义，单位不统一，字长不一致等；然后将原始数据结构作一个从面向应用到面向主题的大转变；还要进行数据综合和计算；数据仓库中的数据综合工作可以在抽取数据时生成，也可以在进入数据仓库以后进行综合时生成。

3. 数据仓库是不可更新的

数据仓库主要是供决策分析之用的，所涉及的数据操作主要是数据查询，一般情况下并不进行修改操作。数据仓库存储的是相当长一段时间内的历史数据，是不同时点数据库快照的集合，以及基于这些快照进行统计、综合和重组的导出数据，不是联机处理的数据。OLTP 数据库中的数据经过抽取(Extracting)、清洗(Cleaning)、转换(Transformation)和装载(Loading)存放到数据仓库中(人们简记这一过程为 ECTL)。一旦数据存放到数据仓库中，数据就不再更新了。

4. 数据仓库是随时间变化的

数据仓库中的数据不可更新，是指数据仓库的用户进行分析处理时是不进行数据更新操作的，但并不是说，在数据仓库的整个生存周期中数据集合是不变的。

数据仓库的数据是随时间的变化不断变化的，这一特征表现在以下三方面。

第一，数据仓库随时间变化不断增加新的数据内容。数据仓库系统必须不断捕捉 OLTP 数据库中新的数据，追加到数据仓库中去。

第二，数据仓库随时间变化不断删去旧的数据内容。数据仓库的数据也有存储期限，一旦超过了这一期限，过期数据就要被删除。只是数据仓库内的数据时限要远远长久于操作型数据的时限。在操作型环境中一般只保存 60～90 天的数据，而在数据仓库中则需要保存较长时限的数据(如 5～20 年)，以适应趋势分析的要求。

第三，数据仓库中包含大量的综合数据，这些综合数据中很多与时间有关，如数据按照某一时间段进行综合，或隔一定的时间片进行采样等，这些数据就会随着时间的变化不断地

进行重新综合。因此，数据仓库数据的码键都包含时间项，以标明数据的历史时期。

二、数据仓库中的数据组织

上面介绍了数据仓库中数据的4个基本特征，下面讲解数据仓库的数据组织结构。

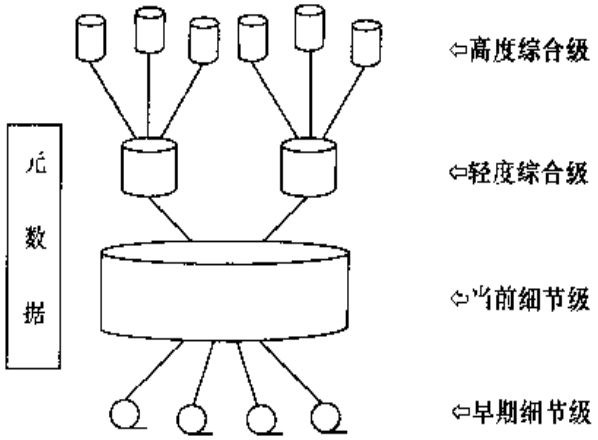


图 17.1 数据仓库的数据组织结构

数据仓库中的数据分为多个级别：早期细节级、当前细节级、轻度综合级、高度综合级。数据仓库的数据组织结构如图 17.1 所示。源数据经过抽取、清洗、转换、装载进入数据仓库。首先进入当前细节级，根据具体的分析处理需求再进行综合。随着时间的推移，早期的数据将转入早期细节级。

数据仓库中数据具有不同的综合级别，一般称之为“粒度”。粒度是数据仓库数据组织的一个重要概念。粒度越大，表示细节程度越低，综合程度越高。不同的数据粒度可以回答不同类型的问题。由于数据仓库的主要应用是分析处理，绝大部分查询都针对综合数据，因而多重粒度的数据组织可以大大提高联机分析的效率。

不同粒度的数据可以存储在不同级别的存储设备上。例如，可以将大粒度数据存储于快速设备甚至放在内存。这样，对于绝大多数查询分析，系统性能将大大提高。而小粒度数据则可存储在磁带磁盘阵列、光盘组或磁带上。

三、数据仓库系统的体系结构

数据仓库系统架构如图 17.2 所示，由数据仓库的后台工具、数据仓库服务器、OLAP 服务器和前台工具组成。

数据仓库的后台工具包括数据抽取、清洗、转换、装载和维护(Maintain)工具。目前许多公司产品把后台工具简记为 ECTL 工具或 ETL 工具。

数据仓库服务器相当于数据库系统中的 DBMS，它负责管理数据仓库中数据的存储管理和数据存取，并给 OLAP 服务器和前台工具提供存取接口(如 SQL 查询接口)。数据仓库服务器目前一般是 RDBMS 或扩展的 RDBMS，即由传统数据库厂商对 DBMS 加以扩展修改，使它能更好地支持数据仓库的功能。

OLAP 服务器透明地为前台工具和用户提供多维数据视图。用户不必关心它的分析数

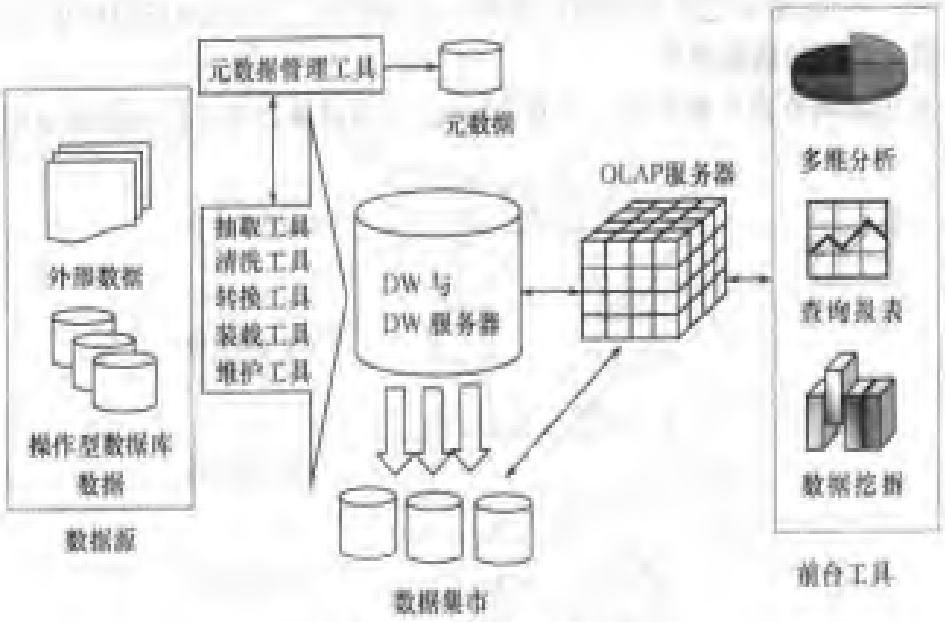


图 17.2 数据仓库体系结构

据(即多维数据)到底存储在什么地方,是怎么存储的。OLAP 服务器则必须考虑物理上这些分析数据的存储问题。

前台工具包括查询报表工具、多维分析工具、数据挖掘工具和分析结果可视化工具等。

17.2 联机分析处理技术

OLAP(On-Line Analytical Processing)即联机分析处理,是以海量数据为基础的复杂分析技术。OLAP 支持各级管理决策人员从不同的角度、快速灵活地对数据仓库中的数据进行复杂查询和多维分析处理,辅助各级领导进行正确决策,提高企业的竞争力。

一、多维数据模型

多维数据模型是数据分析时用户的数据视图,是面向分析的数据模型,用于给分析人员提供多种观察的视角和面向分析的操作。

多维数据模型的数据结构可以用这样来一个多维数组来表示:(维 1,维 2,…,维 n,度量值),例如,图 17.3 所示的电器商品销售数据是按时间、地区、商品种类,加上变量"销售额"组成的一个三维数组(地区,时间,电器商品种类,销售额)。三维数组可以用一个立方体来直观地表示。一般地多维数组用多维立方体 CUBE 来表示。多维立方体 CUBE 也称为超立方体。

二、多维分析操作

常用的 OLAP 多维分析操作有切片(slice)、切块(dice)、旋转(pivot)、向上综合(roll-up)、向下钻取(drill-down)等。通过这些操作,使用户能从多个角度多侧面观察数据,剖析

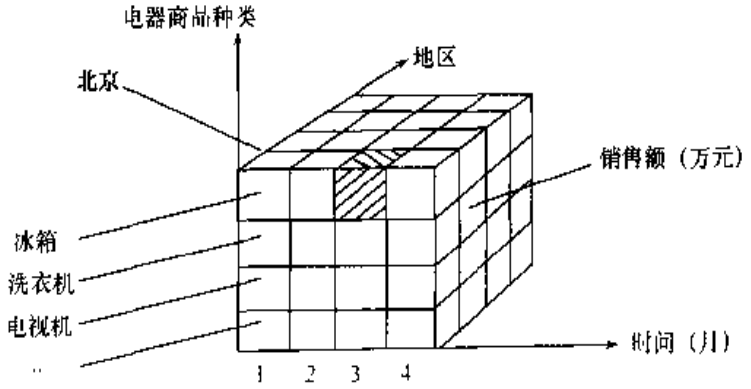


图 17.3 按商品种类、时间和地区组织的电器商品销售数据

数据，从而深入地了解包含在数据中的信息与内涵。

三、OLAP 的实现方式

OLAP 服务器透明地为分析软件和用户提供多维数据视图，实现对多维数据的存储、索引、查询和优化等等。OLAP 服务器一般按照多维数据模型的不同实现方式，有 MOLAP 结构、ROLAP 结构、HOLAP 结构等多种结构。

MOLAP 结构直接以多维立方体 CUBE 来组织数据，以多维数组来存储数据，支持直接对多维数据的各种操作。人们也常常称这种按照多维立方体来组织和存储的数据结构为多维数据库（Multi-Dimension DataBase，简记为 MDDB）。Arbor 公司的 Essbase 是一个 MOLAP 服务器。

ROLAP 结构用 RDBMS 或扩展的 RDBMS 来管理多维数据，用关系的表来组织和存储多维数据。同时，它将多维立方体上的操作映射为标准的关系操作。

ROLAP 将多维立方体结构划分为两类表，一类是事实（fact）表，另一类是维表。事实表用来描述和存储多维立方体的度量值及各个维的码值；维表用来描述维信息。ROLAP 用关系数据库的二维表来表示事实表和维表，也就是说，ROLAP 用“星形模式”和“雪片模式”来表示多维数据模型。

星形模式通常由一个中心表（事实表）和一组维表组成。如图 17.4 所示的星形模式的中心是销售事实表，其周围的维表有时间维表、顾客维表、销售员维表、制造商维表和产品维表。事实表一般很大，维表一般较小。

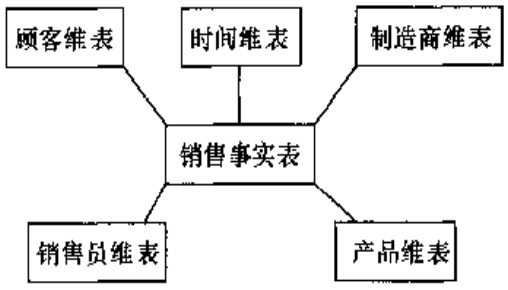


图 17.4 星形模式

雪片模式就是对维表按层次进一步细化后形成的。如图 17.4 的星形模式，顾客维表可以按所在地区位置分类聚集；时间维表则可以有两类层次——日、月，日、周；制造商维表可以按工厂及工厂按所在地区分层等。如图 17.5 所示，在“星形”维表的角上又出现了分支，这样变形的星形模式被称为“雪片模式”（Snow Flake

Schema)。

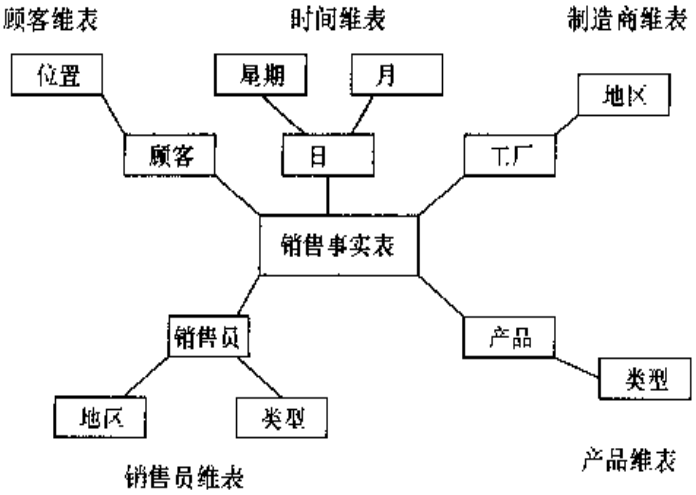


图 17.5　雪片模式

OLAP 软件提供的是多维分析和辅助决策功能。对于深层次的分析和发现数据中隐含的规律和知识，则需要数据挖掘(Data Mining，DM)技术和相应的数据挖掘软件来完成。

17.3　数据挖掘技术

面对日益激烈的市场竞争，客户对迅速应答各种业务问题的能力的要求不断提高，不仅要求回答发生什么，为何发生，还要回答将发生什么。数据挖掘技术正是支持回答“将发生什么”这类业务问题的。

一、数据挖掘的概念

数据挖掘是从大量数据中发现并提取隐藏在内的、人们事先不知道的但又可能有用的信息和知识的一种新技术。

数据挖掘的目的是帮助决策者寻找数据间潜在的关联，发现经营者被忽略的要素，而这些要素对预测趋势、决策行为也许是十分有用的信息。

数据挖掘技术涉及数据库技术、人工智能技术、机器学习、统计分析等多种技术，它使决策支持系统(DSS)跨入了一个新阶段。

二、数据挖掘和传统分析方法的区别

传统的 DSS 系统通常是在某个假设的前提下通过数据查询和分析来验证或否定这个假设。数据挖掘与传统的数据分析(如查询、报表、联机应用分析)的本质区别是数据挖掘是在没有明确假设的前提下去挖掘信息，发现知识。

数据挖掘技术是基于大量的来自实际应用的数据，进行自动分析、归纳推理，从中发掘出数据间潜在的模式，或产生联想，建立新的业务模型帮助决策者调整企业发展策略，进行正确决策。

数据挖掘所得到的信息应具有事先未知、有效和可实用3个特征。

事先未知的信息是指该信息是未曾预料到的，即数据挖掘是要发现那些不能靠直觉发现的信息或知识，甚至是违背直觉的信息或知识。挖掘出的信息越是出乎意料，就可能越有价值。在商业应用中最典型的例子就是，一家连锁店通过数据挖掘，发现了小孩尿布和啤酒之间的销售模式有着惊人的联系。

三、数据挖掘的数据源

数据挖掘的数据主要有两种来源，可以是从数据仓库中来的，也可以是直接从数据库中来。这些实际的应用数据往往是不完全的、有噪声的、模糊的、随机的，因此要根据不同的需求在挖掘之前进行预处理。

从数据仓库中直接得到数据挖掘的数据有许多好处。因为数据仓库的数据已经过了预处理，许多数据不一致的问题都较好地解决了，在数据挖掘时大大减少了清理数据的工作量。

当然为了数据挖掘也不必非得建立一个数据仓库，数据仓库不是必需的。建立一个巨大的数据仓库，要把各个不同源的数据集成在一起，解决所有的数据冲突问题，然后把所有的数据导到一个数据仓库内，是一项巨大的工程，可能要用几年的时间花上百万的经费才能完成。如果只是为了数据挖掘，可以把一个或几个 OLTP 数据库导入一个只读的数据库中，然后在上面进行数据挖掘。

所有的数据还要再次进行选择，具体的选择方式与任务相关。挖掘的结果需要进行评价才能最终成为有用的信息，按照评价结果的不同，数据可能需要反馈到不同的阶段，重新进行分析计算。

四、数据挖掘的功能

数据挖掘的功能主要有以下几种。

(1) 概念描述：就是指归纳总结出数据的某些特征。

(2) 关联分析：若两个或多个变量的取值之间存在某种规律性，就称为关联。包括相关关联和因果关联。关联规则不仅是单维关联，也可能是多维之间的关联。

(3) 分类和预测：找到一定的函数或者模型来描述和区分数据类之间的区别，用这些函数和模型对未来进行预测。这些数据类是事先已经知道的。分类的结果表示为决策树、分类规则或神经网络。

(4) 聚类：将数据分为多个类，使得类内部数据之间的差异最小，而类之间数据的差异最大。与分类不同的是，聚类前并不知道类的个数。聚类技术主要包括传统的模式识别方法和数学分类学等。

(5) 孤立点的检测：孤立点是指数据中的整体表现行为不一致的那些数据集合。这些数据虽然是一些特例，但往往在错误检查和特例分析中是很有用的。

(6) 趋势和演变分析：描述行为随着时间变化的对象所遵循的规律或趋势。

一个典型的数据挖掘系统的体系结构如图 17.6 所示。

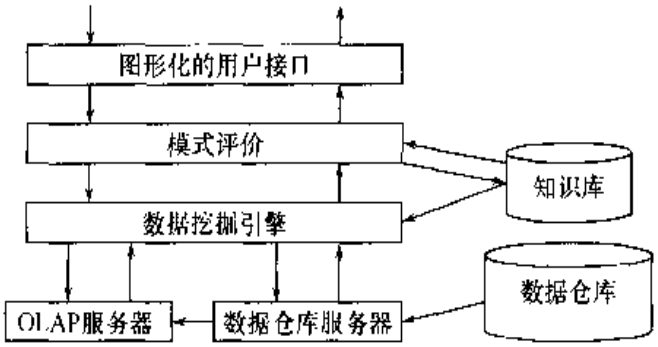


图 17.6　典型的数据挖掘系统的体系结构

在进行挖掘之前首先要明确挖掘的任务，比如说是要进行分类、聚类或寻找关联规则等，然后根据这些任务来对所选择数据进行预处理，之后再选择具体的算法进行挖掘。最后要对挖掘出来的模式进行评价，消减其中重复的部分，将最终的结果展现出来。数据挖掘技术从一开始就是面向应用的，尤其在银行、电信、保险、交通、零售（如超级市场）等商业领域有着极其广泛的应用前景。

17.4　小　　结

数据仓库 DW、OLAP 和数据挖掘 DM 是作为三种独立的信息处理技术出现的。数据仓库用于数据的存储和组织，OLAP 集中于数据的分析，数据挖掘则致力于知识的自动发现。它们都可以分别应用到信息系统的设计和实现中，以提高相应部分的处理能力。由于这三种技术内在的联系性和互补性，将它们结合起来就成为一种新的 DSS 构架，成为商务智能（Business Intelligent，BI）的 3 个支柱。

即：DW + OLAP + DM→支持 BI 的可行方案

这一构架以数据库中的大量数据为基础，其特点如下。

（1）在底层的数据库中保存了大量的事务级细节数据。这些数据是整个 DSS 系统的数据来源。

（2）数据仓库对底层数据库中的事务级数据进行集成、转换、综合，重新组织成面向全局的数据视图，为 DSS 提供数据存储和组织的基础。

（3）OLAP 从数据仓库中的集成数据出发，构建面向分析的多维数据模型，再使用多维分析方法从多个不同的视角对多维数据进行分析、比较，分析活动从以前的方法驱动转向了数据驱动，分析方法和数据结构实现了分离。

（4）数据挖掘以数据仓库和数据库中的大量数据为基础，自动地发现数据中的潜在模式，并以这些模式为基础自动地进行预测分析。

数据挖掘表明知识就隐藏在日常积累下来的大量数据之中，仅靠复杂的算法和推理并不能发现知识，数据才是知识的源泉。

习　题

1. 数据仓库的4个基本特征是什么？
2. 操作型数据和分析型数据的主要区别是什么？
3. 在基于关系数据库的OLAP实现中，举例说明如何利用关系数据库的二维表来表达多维概念。
4. 数据挖掘和传统分析方法的主要不同是什么？

本章参考文献

1 王珊等. 数据仓库技术与联机分析处理. 数据库技术丛书之一. 北京：科学出版社，1998

（本书全面系统地讲解了数据仓库的概念、技术和数据仓库系统建设方法，介绍了以数据仓库为基础的联机分析处理技术及其应用实例。）

2 Inmon W H. 数据仓库. 王志海等译. 北京：机械工业出版社，2000

3 Inmon W H 等. 数据仓库管理. 王天佑译. 北京：电子工业出版社，2000

4 Kimball R 等. 数据仓库工具箱. 谭明金译. 北京：电子工业出版社，2003

5 Imhoff C 等. 数据仓库设计. 于戈等译. 北京：机械工业出版社，2004

6 Mattison R. Web 仓库工程与知识管理. 高军等译. 北京：清华大学出版社，2003

7 Bain T 等. SQL Server 2000 数据仓库与 Analysis Services. 邵勇等译. 北京：中国电力出版社，2003